## Updates to *MPC8536E Reference Manual*, Rev. 1, as of 08/07/2015

This section provides updates to the *MPC8536E Reference Manual*, Rev. 1. We are providing known corrections, but do not guarantee that the list is exhaustive. For convenience, the section number and page number of the item in the reference manual are provided.

**Note:** This PDF file contains updates embedded as inline sticky notes; use the provided links and scroll to the inline location. Future versions of the document will incorporate the sticky notes into the source text of the document. Freescale recommends viewing this file with Adobe Acrobat Reader.

| Section, Page No. | Changes |
|---|---|
| 14.5.2, 14-16 | In Table 14-4, "Module Memory Map," updated the reset values for both eTSEC_ID1 (was 0x0124_0000, now 0x0124_0106) and eTSEC_ID2 (was 0x0030_00F0, now 0x00F8_00F0). |

# MPC8536E PowerQUICC III™ Integrated Processor Reference Manual

**Supports**

MPC8536E

MPC8535E

MPC8536ERM
Rev. 1
05/2009

***How to Reach Us:***

**Home Page:**
www.freescale.com

**email:**
support@freescale.com

**USA/Europe or Locations Not Listed:**
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
(800) 521-6274
480-768-2130
support@freescale.com

**Europe, Middle East, and Africa:**
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

**Japan:**
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku
Tokyo 153-0064, Japan
0120 191014
+81 2666 8080
support.japan@freescale.com

**Asia/Pacific:**
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate,
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

**For Literature Requests Only:**
Freescale Semiconductor
Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
(800) 441-2447
303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor
@hibbertgroup.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters which may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including “Typicals” must be validated for each customer application by customer’s technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

Document Number: MPC8536ERM
Rev. 1, 05/2009

# Contents

**Paragraph Number** | **Title** | **Page Number**

## About This Book

Audience ....................................................................... cix
Organizations ................................................................. cix
Suggested Reading ......................................................... cxii
General Information ....................................................... cxii
Related Documentation ................................................... cxii
Conventions ................................................................... cxiii
Signal Conventions ......................................................... cxiv
Register Access Conventions .......................................... cxiv
Acronyms and Abbreviations ........................................... cxiv

## Part I
## Overview

## Chapter 1
## Overview

1.1 Introduction ........................................................ 1-1
1.2 MPC8536E Overview ............................................ 1-2
1.2.1 Key Features ................................................... 1-2
1.3 MPC8536E Architecture Overview ........................ 1-3
1.3.1 e500 Core and Memory Unit ............................... 1-3
1.3.2 e500 Coherency Module (ECM) and Address Map ... 1-3
1.3.3 Integrated Security Engine (SEC) ......................... 1-4
1.3.4 High-Speed Interface Blocks (SerDes) ................... 1-4
1.3.4.1 Eight-Lane SerDes ........................................... 1-5
1.3.4.2 Two-Lane SerDes ............................................. 1-5
1.3.5 Enhanced Three-Speed Ethernet Controllers (eTSEC) ... 1-5
1.3.6 Universal Serial Bus (USB) Dual-Role Controllers ... 1-6
1.3.6.1 Host Mode Operation ....................................... 1-6
1.3.6.2 Device Mode Operation ..................................... 1-6
1.3.7 DDR SDRAM Controller ..................................... 1-7
1.3.8 Power Management Controller ........................... 1-7
1.3.9 PCI Express Controller ...................................... 1-8
1.3.10 Programmable Interrupt Controller (PIC) ............ 1-8
1.3.11 Enhanced Secure Digital Host Controller (eSDHC) ... 1-8

# Contents

Paragraph Number — Title — Page Number

1.3.12 eSPI Interface ........ 1-8
1.3.13 Serial ATA (SATA) Controllers ........ 1-9
1.3.14 DMA Controller, I$^2$C, DUART, eLBC ........ 1-9
1.4 MPC8536E Application Examples ........ 1-11
1.4.1 Multi-Function Printer ........ 1-11
1.4.2 Network Attached Storage ........ 1-12
1.4.3 Gaming Kiosk ........ 1-13
1.4.4 Network Controller ........ 1-14

## Chapter 2
## Memory Map

2.1 Local Memory Map Overview and Example ........ 2-1
2.2 Address Translation and Mapping ........ 2-3
2.2.1 SRAM Windows ........ 2-4
2.2.2 Window into Configuration Space ........ 2-4
2.2.3 Local Access Windows ........ 2-4
2.2.3.1 Local Access Register Memory Map ........ 2-5
2.2.3.2 Local Access IP Block Revision Register 1 (LAIPBRR1) ........ 2-6
2.2.3.3 Local Access IP Block Revision Register 2 (LAIPBRR2) ........ 2-6
2.2.3.4 Local Access Window *n* Base Address Registers (LAWBAR0–LAWBAR9) ........ 2-7
2.2.3.5 Local Access Window *n* Attributes Registers (LAWAR0–LAWAR9) ........ 2-7
2.2.3.6 Precedence of Local Access Windows ........ 2-8
2.2.3.7 Configuring Local Access Windows ........ 2-8
2.2.3.8 Distinguishing Local Access Windows from Other Mapping Functions ........ 2-9
2.2.3.9 Illegal Interaction Between Local Access Windows and DDR SDRAM Chip Selects ........ 2-9
2.2.4 Outbound Address Translation and Mapping Windows ........ 2-9
2.2.5 Inbound Address Translation and Mapping Windows ........ 2-10
2.2.5.1 PCI Inbound ATMU ........ 2-10
2.2.5.2 PCI Express Inbound ATMU ........ 2-10
2.2.5.3 Illegal Interaction Between Inbound ATMUs and Local Access Windows ........ 2-10
2.3 Configuration, Control, and Status Register Map ........ 2-10
2.3.1 Accessing CCSR Memory from the Local Processor ........ 2-11
2.3.2 Accessing CCSR Memory from External Masters ........ 2-11
2.3.3 Organization of CCSR Memory ........ 2-11
2.3.4 General Utilities Registers ........ 2-12
2.3.5 Interrupt Controller and CCSR ........ 2-13
2.3.6 Device-Specific Utilities ........ 2-13
2.4 Complete CCSR Map ........ 2-14

# Contents

Paragraph Number | Title | Page Number

## Chapter 3
## Signal Descriptions

3.1 Signals Overview ... 3-1
3.2 Configuration Signals Sampled at Reset ... 3-15
3.3 Output Signal States During Reset ... 3-17

## Chapter 4
## Reset, Clocking, and Initialization

4.1 Overview ... 4-1
4.2 External Signal Descriptions ... 4-1
4.2.1 System Control Signals ... 4-2
4.2.2 Clock Signals ... 4-3
4.3 Memory Map/Register Definition ... 4-3
4.3.1 Local Configuration Control ... 4-4
4.3.1.1 Accessing Configuration, Control, and Status Registers ... 4-4
4.3.1.1.1 Updating CCSRBAR ... 4-4
4.3.1.1.2 Configuration, Control, and Status Registers Base Address Register (CCSRBAR) ... 4-5
4.3.1.2 Accessing Alternate Configuration Space ... 4-5
4.3.1.2.1 Alternate Configuration Base Address Register (ALTCBAR) ... 4-6
4.3.1.2.2 Alternate Configuration Attribute Register (ALTCAR) ... 4-6
4.3.1.3 Boot Page Translation ... 4-7
4.3.1.3.1 Boot Page Translation Register (BPTR) ... 4-7
4.3.2 Boot Sequencer ... 4-8
4.4 Functional Description ... 4-8
4.4.1 Reset Operations ... 4-8
4.4.1.1 Soft Reset ... 4-8
4.4.1.2 Hard Reset ... 4-8
4.4.2 Power-On Reset Sequence ... 4-9
4.4.3 Power-On Reset Configuration ... 4-11
4.4.3.1 System PLL Ratio ... 4-11
4.4.3.2 e500 Core PLL Ratio ... 4-12
4.4.3.3 DDR PLL Ratio Configuration ... 4-13
4.4.3.4 System Speed Configuration ... 4-13
4.4.3.5 Core Speed Configuration ... 4-14
4.4.3.6 Boot ROM Location ... 4-14
4.4.3.7 Host/Agent Configuration ... 4-15
4.4.3.8 SerDes1 I/O Port Selection ... 4-16
4.4.3.9 SerDes2 I/O Port Selection ... 4-17

# Contents

| Paragraph Number | Title | Page Number |
|---|---|---|
| 4.4.3.10 | CPU Boot Configuration | 4-17 |
| 4.4.3.11 | Boot Sequencer Configuration | 4-18 |
| 4.4.3.12 | DDR SDRAM Type | 4-19 |
| 4.4.3.13 | Serdes 2 Reference Clock Configuration | 4-19 |
| 4.4.3.14 | eTSEC1 width | 4-20 |
| 4.4.3.15 | eTSEC3 Width | 4-20 |
| 4.4.3.16 | eTSEC1 Protocol | 4-21 |
| 4.4.3.17 | eTSEC3 Protocol | 4-21 |
| 4.4.3.18 | PCI Clock Selection | 4-22 |
| 4.4.3.19 | PCI Speed Configuration | 4-22 |
| 4.4.3.20 | PCI I/O Impedance | 4-22 |
| 4.4.3.21 | PCI Arbiter Configuration | 4-23 |
| 4.4.3.22 | Memory Debug Configuration | 4-23 |
| 4.4.3.23 | DDR Debug Configuration | 4-23 |
| 4.4.3.24 | General-Purpose POR Configuration | 4-24 |
| 4.4.3.25 | Engineering Use POR Configuration | 4-24 |
| 4.4.4 | Clocking | 4-24 |
| 4.4.4.1 | System Clock/PCI Clock/DDR Clock | 4-24 |
| 4.4.4.2 | PCI Express and SGMII Clocks | 4-25 |
| 4.4.4.2.1 | Minimum Frequency Requirements | 4-26 |
| 4.4.4.3 | Ethernet Clocks | 4-26 |
| 4.4.4.4 | Real Time Clock | 4-26 |
| 4.5 | Initialization/Applications Information | 4-27 |
| 4.5.1 | System Boot | 4-27 |
| 4.5.1.1 | eSDHC Boot | 4-27 |
| 4.5.1.1.1 | Overview | 4-27 |
| 4.5.1.1.2 | Features | 4-28 |
| 4.5.1.1.3 | SD/MMC Card Data Structure | 4-29 |
| 4.5.1.1.4 | eSDHC Controller Initial Configuration | 4-33 |
| 4.5.1.1.5 | eSDHC Controller Boot Sequence | 4-33 |
| 4.5.1.1.6 | eSDHC Boot Error Handling | 4-34 |
| 4.5.1.2 | eSPI Boot ROM | 4-35 |
| 4.5.1.2.1 | Overview | 4-35 |
| 4.5.1.2.2 | Features | 4-36 |
| 4.5.1.2.3 | EEPROM Data Structure | 4-36 |
| 4.5.1.2.4 | eSPI Controller Configuration | 4-40 |
| 4.5.1.3 | Default e500 Addressing During System Boot | 4-41 |

# Contents

Paragraph Number | Title | Page Number

## Part II
## e500 Core Complex and L2 Cache

## Chapter 5
## e500 Core Integration Details

5.1 e500 Core Overview .......... 5-1
5.2 e500 Core Integration and the Core Complex Bus (CCB) .......... 5-3
5.3 Summary of Core Integratation Details .......... 5-4
5.3.1 Processor Version Register (PVR) and System Version Register (SVR) .......... 5-6

## Chapter 6
## L2 Look-Aside Cache/SRAM

6.1 L2 Cache Overview .......... 6-1
6.1.1 L2 Cache and SRAM Features .......... 6-2
6.2 L2 Cache and SRAM Organization .......... 6-4
6.2.1 Accessing the On-Chip Array as an L2 Cache .......... 6-5
6.2.2 Accessing the On-Chip Array as an SRAM .......... 6-5
6.2.3 Connection of the On-Chip Memory to the System .......... 6-7
6.3 Memory Map/Register Definition .......... 6-8
6.3.1 L2/SRAM Register Descriptions .......... 6-10
6.3.1.1 L2 Control Register (L2CTL) .......... 6-10
6.3.1.2 L2 Cache External Write Registers .......... 6-13
6.3.1.2.1 L2 Cache External Write Address Registers 0–3 (L2CEWAR*n*) .......... 6-13
6.3.1.2.2 L2 Cache External Write Address Registers Extended Address 0–3 (L2CEWAREA*n*) .......... 6-14
6.3.1.2.3 L2 Cache External Write Control Registers 0–3 (L2CEWCR*n*) .......... 6-14
6.3.1.3 L2 Memory-Mapped SRAM Registers .......... 6-15
6.3.1.3.1 L2 Memory-Mapped SRAM Base Address Registers 0–1 (L2SRBAR*n*) .......... 6-16
6.3.1.3.2 L2 Memory-Mapped SRAM Base Address Registers Extended Address 0–1 (L2SRBAREA*n*) .......... 6-17
6.3.1.4 L2 Error Registers .......... 6-17
6.3.1.4.1 Error Injection Registers .......... 6-18
6.3.1.4.2 Error Control and Capture Registers .......... 6-20
6.4 External Writes to the L2 Cache (Cache Stashing) .......... 6-25
6.4.1 Stash-Only Cache Regions .......... 6-26
6.5 L2 Cache Timing .......... 6-27
6.6 L2 Cache and SRAM Coherency .......... 6-27
6.6.1 L2 Cache Coherency Rules .......... 6-28
6.6.2 Memory-Mapped SRAM Coherency Rules .......... 6-29

# Contents

Paragraph Number | Title | Page Number

6.7 L2 Cache Locking ... 6-29
6.7.1 Locking the Entire L2 Cache ... 6-29
6.7.2 Locking Programmed Memory Ranges ... 6-30
6.7.3 Locking Selected Lines ... 6-30
6.7.4 Clearing Locks on Selected Lines ... 6-30
6.7.5 Flash Clearing of Instruction and Data Locks ... 6-31
6.7.6 Locks with Stale Data ... 6-31
6.8 PLRU L2 Replacement Policy ... 6-31
6.8.1 PLRU Bit Update Considerations ... 6-32
6.8.2 Allocation of Lines ... 6-32
6.9 L2 Cache Operation ... 6-33
6.9.1 Initialization ... 6-33
6.9.1.1 L2 Cache Initialization ... 6-33
6.9.1.2 Memory-Mapped SRAM Initialization ... 6-33
6.9.2 Flash Invalidation of the L2 Cache ... 6-34
6.9.3 Managing Errors ... 6-34
6.9.3.1 ECC Errors ... 6-34
6.9.3.2 Tag Parity Errors ... 6-34
6.9.4 L2 Cache States ... 6-34
6.9.5 L2 State Transitions ... 6-35
6.9.6 Error Checking and Correcting (ECC) ... 6-39

## Part III
## Memory, Security, and I/O Interfaces

## Chapter 7
## e500 Coherency Module

7.1 Introduction ... 7-1
7.1.1 Overview ... 7-2
7.1.2 Features ... 7-2
7.2 Memory Map/Register Definition ... 7-3
7.2.1 Register Descriptions ... 7-3
7.2.1.1 ECM CCB Address Configuration Register (EEBACR) ... 7-3
7.2.1.2 ECM CCB Port Configuration Register (EEBPCR) ... 7-4
7.2.1.3 ECM IP Block Revision Register 1 (EIPBRR1) ... 7-5
7.2.1.4 ECM IP Block Revision Register 2 (EIPBRR2) ... 7-5
7.2.1.5 ECM Error Detect Register (EEDR) ... 7-6
7.2.1.6 ECM Error Enable Register (EEER) ... 7-7
7.2.1.7 ECM Error Attributes Capture Register (EEATR) ... 7-7
7.2.1.8 ECM Error Low Address Capture Register (EELADR) ... 7-8

# Contents

Paragraph Number | Title | Page Number

7.2.1.9 ECM Error High Address Capture Register (EEHADR) ....... 7-9
7.3 Functional Description ....... 7-9
7.3.1 I/O Arbiter ....... 7-9
7.3.2 CCB Arbiter ....... 7-9
7.3.3 Transaction Queue ....... 7-10
7.3.4 Global Data Multiplexor ....... 7-10
7.3.5 CCB Interface ....... 7-10
7.4 Initialization/Application Information ....... 7-10

## Chapter 8
## DDR Memory Controller

8.1 Introduction ....... 8-1
8.2 Features ....... 8-2
8.2.1 Modes of Operation ....... 8-3
8.3 External Signal Descriptions ....... 8-3
8.3.1 Signals Overview ....... 8-3
8.3.2 Detailed Signal Descriptions ....... 8-6
8.3.2.1 Memory Interface Signals ....... 8-6
8.3.2.2 Clock Interface Signals ....... 8-10
8.3.2.3 Debug Signals ....... 8-10
8.4 Memory Map/Register Definition ....... 8-10
8.4.1 Register Descriptions ....... 8-12
8.4.1.1 Chip Select Memory Bounds (CS*n*_BNDS) ....... 8-12
8.4.1.2 Chip Select Configuration (CS*n*_CONFIG) ....... 8-13
8.4.1.3 Chip Select Configuration 2 (CSn_CONFIG_2) ....... 8-15
8.4.1.4 DDR SDRAM Timing Configuration 3 (TIMING_CFG_3) ....... 8-16
8.4.1.5 DDR SDRAM Timing Configuration 0 (TIMING_CFG_0) ....... 8-17
8.4.1.6 DDR SDRAM Timing Configuration 1 (TIMING_CFG_1) ....... 8-19
8.4.1.7 DDR SDRAM Timing Configuration 2 (TIMING_CFG_2) ....... 8-21
8.4.1.8 DDR SDRAM Control Configuration (DDR_SDRAM_CFG) ....... 8-23
8.4.1.9 DDR SDRAM Control Configuration 2 (DDR_SDRAM_CFG_2) ....... 8-26
8.4.1.10 DDR SDRAM Mode Configuration (DDR_SDRAM_MODE) ....... 8-29
8.4.1.11 DDR SDRAM Mode 2 Configuration (DDR_SDRAM_MODE_2) ....... 8-29
8.4.1.12 DDR SDRAM Mode Control Register (DDR_SDRAM_MD_CNTL) ....... 8-30
8.4.1.13 DDR SDRAM Interval Configuration (DDR_SDRAM_INTERVAL) ....... 8-33
8.4.1.14 DDR SDRAM Data Initialization (DDR_DATA_INIT) ....... 8-33
8.4.1.15 DDR SDRAM Clock Control (DDR_SDRAM_CLK_CNTL) ....... 8-34
8.4.1.16 DDR Initialization Address (DDR_INIT_ADDR) ....... 8-34
8.4.1.17 DDR Initialization Enable Extended Address (DDR_INIT_EXT_ADDR) ....... 8-35
8.4.1.18 DDR SDRAM Timing Configuration 4 (TIMING_CFG_4) ....... 8-36

# Contents

Paragraph Number | Title | Page Number

8.4.1.19 DDR SDRAM Timing Configuration 5 (TIMING_CFG_5) ................................ 8-37
8.4.1.20 DDR ZQ Calibration Control (DDR_ZQ_CNTL) ................................ 8-39
8.4.1.21 DDR Write Leveling Control (DDR_WRLVL_CNTL) ................................ 8-40
8.4.1.22 DDR Self Refresh Counter (DDR_SR_CNTR) ................................ 8-43
8.4.1.23 DDR SDRAM Register Control Word 1 (DDR_SDRAM_RCW_1) ................................ 8-44
8.4.1.24 DDR SDRAM Register Control Word 2 (DDR_SDRAM_RCW_2) ................................ 8-45
8.4.1.25 DDR Debug Status Register 1 (DDRDSR_1) ................................ 8-46
8.4.1.26 DDR Debug Status Register 2 (DDRDSR_2) ................................ 8-47
8.4.1.27 DDR Control Driver Register 1 (DDRCDR_1) ................................ 8-47
8.4.1.28 DDR Control Driver Register 2 (DDRCDR_2) ................................ 8-50
8.4.1.29 DDR IP Block Revision 1 (DDR_IP_REV1) ................................ 8-50
8.4.1.30 DDR IP Block Revision 2 (DDR_IP_REV2) ................................ 8-51
8.4.1.31 Memory Data Path Error Injection Mask High (DATA_ERR_INJECT_HI) ........ 8-51
8.4.1.32 Memory Data Path Error Injection Mask Low (DATA_ERR_INJECT_LO) ........ 8-52
8.4.1.33 Memory Data Path Error Injection Mask ECC (ERR_INJECT) ................................ 8-52
8.4.1.34 Memory Data Path Read Capture High (CAPTURE_DATA_HI) ................................ 8-53
8.4.1.35 Memory Data Path Read Capture Low (CAPTURE_DATA_LO) ................................ 8-54
8.4.1.36 Memory Data Path Read Capture ECC (CAPTURE_ECC) ................................ 8-54
8.4.1.37 Memory Error Detect (ERR_DETECT) ................................ 8-54
8.4.1.38 Memory Error Disable (ERR_DISABLE) ................................ 8-56
8.4.1.39 Memory Error Interrupt Enable (ERR_INT_EN) ................................ 8-57
8.4.1.40 Memory Error Attributes Capture (CAPTURE_ATTRIBUTES) ................................ 8-58
8.4.1.41 Memory Error Address Capture (CAPTURE_ADDRESS) ................................ 8-58
8.4.1.42 Memory Error Extended Address Capture (CAPTURE_EXT_ADDRESS) ........ 8-59
8.4.1.43 Single-Bit ECC Memory Error Management (ERR_SBE) ................................ 8-59
8.5 Functional Description ................................ 8-60
8.5.1 DDR SDRAM Interface Operation ................................ 8-64
8.5.1.1 Supported DDR SDRAM Organizations ................................ 8-64
8.5.2 DDR SDRAM Address Multiplexing ................................ 8-66
8.5.3 JEDEC Standard DDR SDRAM Interface Commands ................................ 8-71
8.5.4 DDR SDRAM Interface Timing ................................ 8-72
8.5.4.1 Clock Distribution ................................ 8-76
8.5.5 DDR SDRAM Mode-Set Command Timing ................................ 8-76
8.5.6 DDR SDRAM Registered DIMM Mode ................................ 8-77
8.5.7 DDR SDRAM Write Timing Adjustments ................................ 8-78
8.5.8 DDR SDRAM Refresh ................................ 8-79
8.5.8.1 DDR SDRAM Refresh Timing ................................ 8-80
8.5.8.2 DDR SDRAM Refresh and Power-Saving Modes ................................ 8-80
8.5.8.2.1 Self-Refresh in Sleep Mode ................................ 8-82
8.5.9 DDR Data Beat Ordering ................................ 8-83
8.5.10 Page Mode and Logical Bank Retention ................................ 8-83

# Contents

Paragraph Number | Title | Page Number

8.5.11 Error Checking and Correcting (ECC) .... 8-84
8.5.12 Error Management .... 8-86
8.6 Initialization/Application Information .... 8-87
8.6.1 Programming Differences between Memory Types .... 8-89
8.6.2 DDR SDRAM Initialization Sequence .... 8-94
8.6.3 Using Forced Self-Refresh Mode to Implement a Battery-Backed RAM System .... 8-94
8.6.3.1 Hardware Based Self-Refresh .... 8-94
8.6.3.2 Software Based Self-Refresh .... 8-95
8.6.3.3 Bypassing Re-initialization During Battery-Backed Operation .... 8-95

## Chapter 9
## Programmable Interrupt Controller (PIC)

9.1 Introduction .... 9-1
9.1.1 Overview .... 9-1
9.1.2 Interrupts to the Processor Core .... 9-4
9.1.3 Modes of Operation .... 9-4
9.1.3.1 Mixed Mode (GCR[M] = 1) .... 9-5
9.1.3.2 Pass-Through Mode (GCR[M] = 0) .... 9-5
9.1.4 Interrupt Sources .... 9-5
9.1.4.1 Interrupt Routing—Mixed Mode .... 9-6
9.1.4.2 Interrupt Destinations .... 9-6
9.1.4.3 Internal Interrupt Sources .... 9-6
9.2 External Signal Descriptions .... 9-7
9.2.1 Signal Overview .... 9-8
9.2.2 Detailed Signal Descriptions .... 9-8
9.3 Memory Map/Register Definition .... 9-9
9.3.1 Global Registers .... 9-19
9.3.1.1 Block Revision Register 1 (BRR1) .... 9-19
9.3.1.2 Block Revision Register 2 (BRR2) .... 9-19
9.3.1.3 Feature Reporting Register (FRR) .... 9-20
9.3.1.4 Global Configuration Register (GCR) .... 9-21
9.3.1.5 Vendor Identification Register (VIR) .... 9-21
9.3.1.6 Processor Core Initialization Register (PIR) .... 9-22
9.3.1.7 Interprocessor Interrupt Vector/Priority Registers (IPIVPR0–IPIVPR3) .... 9-22
9.3.1.8 Spurious Vector Register (SVR) .... 9-23
9.3.2 Global Timer Registers .... 9-23
9.3.2.1 Timer Frequency Reporting Register (TFRRA–TFRRB) .... 9-24
9.3.2.2 Global Timer Current Count Registers (GTCCRA0–GTCCRA3, GTCCRB0–GTCCRB3) .... 9-24

# Contents

**Paragraph Number** **Title** **Page Number**

9.3.2.3 Global Timer Base Count Registers (GTBCRA0–GTBCRA3, GTBCRB0–GTBCRB3)........ 9-25
9.3.2.4 Global Timer Vector/Priority Registers (GTVPRA0–GTVPRA3, GTVPRB0–GTVPRB3)........ 9-25
9.3.2.5 Global Timer Destination Registers (GTDRA0–GTDRA3, GTDRB0–GTDRB3) 9-26
9.3.2.6 Timer Control Registers (TCRA–TCRB)........ 9-27
9.3.3 IRQ_OUT and Critical Interrupt Summary Registers........ 9-28
9.3.3.1 External Interrupt Summary Register (ERQSR)........ 9-29
9.3.3.2 IRQ_OUT Summary Register 0 (IRQSR0)........ 9-29
9.3.3.3 IRQ_OUT Summary Register 1 (IRQSR1)........ 9-30
9.3.3.4 IRQ_OUT Summary Register 2 (IRQSR2)........ 9-31
9.3.3.5 Critical Interrupt Summary Register 0 (CISR0)........ 9-31
9.3.3.6 Critical Interrupt Summary Register 1 (CISR1)........ 9-32
9.3.3.7 Critical Interrupt Summary Register 2 (CISR2)........ 9-32
9.3.4 Performance Monitor Mask Registers (PMMRs)........ 9-32
9.3.4.1 Performance Monitor Mask Registers 0 (PM0MR0–PM3MR0)........ 9-33
9.3.4.2 Performance Monitor Mask Registers 1 (PM0MR1–PM3MR1)........ 9-34
9.3.4.3 Performance Monitor Mask Registers 2 (PM0MR2–PM3MR2)........ 9-34
9.3.5 Message Registers........ 9-34
9.3.5.1 Message Registers (MSGR0–MSGR7)........ 9-35
9.3.5.2 Message Enable Register (MER)........ 9-35
9.3.5.3 Message Status Register (MSR)........ 9-36
9.3.6 Shared Message Signaled Registers........ 9-36
9.3.6.1 Shared Message Signaled Interrupt Registers (MSIR0–MSIR7)........ 9-37
9.3.6.2 Shared Message Signaled Interrupt Status Register (MSISR)........ 9-37
9.3.6.3 Shared Message Signaled Interrupt Index Register (MSIIR)........ 9-38
9.3.6.4 Shared Message Signaled Interrupt Vector/Priority Register (MSIVPRs)........ 9-38
9.3.6.5 Shared Message Signaled Interrupt Destination Registers 0–7 (MSIDR*n*)........ 9-39
9.3.7 Interrupt Source Configuration Registers........ 9-40
9.3.7.1 External Interrupt Vector/Priority Registers (EIVPR0–EIVPR11)........ 9-41
9.3.7.2 External Interrupt Destination Registers (EIDR0–EIDR11)........ 9-42
9.3.7.3 Internal Interrupt Vector/Priority Registers (IIVPRn)........ 9-43
9.3.7.4 Internal Interrupt Destination Registers (IIDRn)........ 9-44
9.3.7.5 Messaging Interrupt Vector/Priority Registers (MIVPRn)........ 9-45
9.3.7.6 Messaging Interrupt Destination Registers (MIDR0–MIDR7)........ 9-46
9.3.8 Per-CPU (Private Access) Registers........ 9-46
9.3.8.1 Interprocessor Interrupt Dispatch Register (IPIDR0–IPIDR3)........ 9-48
9.3.8.2 Processor Core Current Task Priority Registers 0–1 (CTPR0–CTPR1)........ 9-49
9.3.8.3 Who Am I Registers 0–1 (WHOAMI0–WHOAMI1)........ 9-50
9.3.8.4 Processor Core Interrupt Acknowledge Registers 0–1 (IACK0–IACK1)........ 9-50
9.3.8.5 Processor Core End of Interrupt Registers (EOI0–EOI1)........ 9-51

# Contents

Paragraph Number | Title | Page Number

9.4 Functional Description ........ 9-52

9.4.1 Flow of Interrupt Control ........ 9-52

9.4.1.1 Interrupts Routed to $\overline{\text{cint}}$ or $\overline{\text{IRQ\_OUT}}$ ........ 9-52

9.4.1.2 Interrupts Routed to *int* ........ 9-52

9.4.1.2.1 Interrupt Source Priority ........ 9-54

9.4.1.2.2 Interrupt Acknowledge ........ 9-54

9.4.1.2.3 Spurious Vector Generation ........ 9-55

9.4.1.2.4 Nesting of Interrupts ........ 9-55

9.4.2 Interprocessor Interrupts ........ 9-55

9.4.3 Message Interrupts ........ 9-56

9.4.4 Shared Message Signaled Interrupts ........ 9-56

9.4.5 PCI Express INTx/IRQ*n* Sharing ........ 9-56

9.4.6 Global Timers ........ 9-57

9.4.7 Resets ........ 9-57

9.4.8 Resetting the PIC ........ 9-58

9.4.8.1 Processor Core Initialization ........ 9-58

9.5 Initialization/Application Information ........ 9-58

9.5.1 Programming Guidelines ........ 9-58

9.5.1.1 PIC Registers ........ 9-58

9.5.1.2 Changing Interrupt Source Configuration ........ 9-60

## Chapter 10
## Security Engine (SEC) 3.0

10.1 SEC Architecture Overview ........ 10-3

10.1.1 Descriptor Overview ........ 10-4

10.1.2 Polychannel Overview ........ 10-5

10.1.3 Controller Overview ........ 10-6

10.1.4 Execution Units (EUs) Overview ........ 10-7

10.1.4.1 Public Key Execution Unit (PKEU) ........ 10-7

10.1.4.1.1 Elliptic Curve Operations ........ 10-7

10.1.4.1.2 Modular Exponentiation Operations ........ 10-8

10.1.4.2 Data Encryption Standard Execution Unit (DEU) ........ 10-8

10.1.4.3 Advanced Encryption Standard Execution Unit (AESU) ........ 10-9

10.1.4.4 Arc Four Execution Unit (AFEU) ........ 10-10

10.1.4.5 Message Digest Execution Unit (MDEU) ........ 10-10

10.1.4.6 Kasumi Execution Unit (KEU) ........ 10-10

10.1.4.7 Cyclical Redundancy Check Unit (CRCU) ........ 10-10

10.1.4.8 Random Number Generator Unit (RNGU) ........ 10-11

10.2 Configuration of Internal Memory Space ........ 10-11

10.3 Descriptors ........ 10-19

# Contents

| Paragraph Number | Title | Page Number |
|---|---|---|

10.3.1 Descriptor Structure ........ 10-20
10.3.2 Descriptor Format: Header Dword ........ 10-20
10.3.2.1 Selecting Execution Units—EU_SEL0 and EU_SEL1 ........ 10-22
10.3.2.2 Selecting Descriptor Type—DESC_TYPE ........ 10-23
10.3.3 Descriptor Format: Pointer Dwords ........ 10-25
10.3.4 Link Table Format ........ 10-26
10.3.4.1 Example of Link Table Operation ........ 10-29
10.3.5 Descriptor Types ........ 10-30
10.4 Polychannel ........ 10-32
10.4.1 Channel Operation ........ 10-32
10.4.1.1 Channel Descriptor Processing ........ 10-32
10.4.1.2 Channel Arbitration ........ 10-33
10.4.1.3 Channel Host Notification ........ 10-34
10.4.2 Channel Interrupts ........ 10-34
10.4.2.1 Channel Done Interrupt ........ 10-35
10.4.2.2 Channel Error Interrupt ........ 10-35
10.4.3 Polychannel Registers ........ 10-35
10.4.3.1 Traffic Counters ........ 10-35
10.4.3.1.1 Fetch FIFO Enqueue Counter ........ 10-35
10.4.3.1.2 Descriptor Finished Counter ........ 10-36
10.4.3.1.3 Data Bytes In Counter ........ 10-36
10.4.3.1.4 Data Bytes Out Counter ........ 10-37
10.4.4 Channel Registers ........ 10-37
10.4.4.1 Channel Configuration Register (CCR) ........ 10-37
10.4.4.2 Channel Status Register (CSR) ........ 10-41
10.4.4.3 Current Descriptor Pointer Register (CDPR) ........ 10-43
10.4.4.4 Fetch FIFO Enqueue Register (FFER) ........ 10-44
10.4.5 Channel Buffers and Tables ........ 10-45
10.4.5.1 Descriptor Buffer (DB) ........ 10-45
10.4.5.2 Scatter and Gather Link Tables (SLT, GLT) ........ 10-45
10.5 Controller ........ 10-46
10.5.1 Bus Transfers ........ 10-46
10.5.1.1 Host-Controlled Access ........ 10-46
10.5.1.2 Channel-Controlled Access ........ 10-47
10.5.1.2.1 Channel Controlled Read—Detailed Description ........ 10-47
10.5.1.2.2 System Bus Master Write—Detailed Description ........ 10-48
10.5.2 Arbitration Algorithms ........ 10-48
10.5.2.1 Round-Robin Arbitration ........ 10-48
10.5.2.2 Weighted Priority Arbitration ........ 10-48
10.5.3 Controller Interrupts ........ 10-49
10.5.3.1 Controller Interrupt Conditions and Interrupt Generation ........ 10-49

# Contents

| Paragraph Number | Title | Page Number |
|---|---|---|
| 10.5.3.2 | Blocking of Interrupts | 10-49 |
| 10.5.3.3 | Interrupt Handling | 10-49 |
| 10.5.4 | Controller Registers | 10-50 |
| 10.5.4.1 | EU Assignment Status Register (EUASR) | 10-50 |
| 10.5.4.2 | Interrupt Enable, Interrupt Status, and Interrupt Clear Registers (IER, ISR, ICR) | 10-50 |
| 10.5.4.2.1 | Interrupt Enable Register (IER) | 10-53 |
| 10.5.4.2.2 | Interrupt Status Register (ISR) | 10-53 |
| 10.5.4.3 | Interrupt Clear Register (ICR) | 10-54 |
| 10.5.4.4 | ID Register | 10-54 |
| 10.5.4.5 | IP Block Revision Register | 10-54 |
| 10.5.4.6 | Master Control Register (MCR) | 10-55 |
| 10.6 | Power Saving Mode | 10-57 |
| 10.7 | Execution Units | 10-57 |
| 10.7.1 | Advanced Encryption Standard Execution Unit (AESU) | 10-57 |
| 10.7.1.1 | ICV Checking in AESU | 10-58 |
| 10.7.1.2 | AESU Mode Register | 10-58 |
| 10.7.1.3 | AESU Key Size Register | 10-61 |
| 10.7.1.4 | AESU Data Size Register | 10-61 |
| 10.7.1.5 | AESU Reset Control Register | 10-62 |
| 10.7.1.6 | AESU Status Register | 10-62 |
| 10.7.1.7 | AESU Interrupt Status Register | 10-64 |
| 10.7.1.8 | AESU Interrupt Mask Register | 10-66 |
| 10.7.1.9 | AESU ICV Size Register | 10-67 |
| 10.7.1.10 | AESU End of Message Register | 10-68 |
| 10.7.1.11 | AESU Context Registers | 10-68 |
| 10.7.1.11.1 | Context for Confidentiality Cipher Modes | 10-68 |
| 10.7.1.11.2 | Context for Data Integrity Cipher Modes | 10-71 |
| 10.7.1.11.3 | Context for Confidentiality and Data Integrity Cipher Modes | 10-74 |
| 10.7.1.12 | AESU Key Registers | 10-87 |
| 10.7.1.12.1 | AESU FIFOs | 10-88 |
| 10.7.2 | ARC4 Execution Unit (AFEU) | 10-88 |
| 10.7.2.1 | AFEU Mode Register | 10-89 |
| 10.7.2.2 | AFEU Key Size Register | 10-89 |
| 10.7.2.3 | AFEU Context/Data Size Register | 10-90 |
| 10.7.2.4 | AFEU Reset Control Register | 10-91 |
| 10.7.2.5 | AFEU Status Register | 10-91 |
| 10.7.2.6 | AFEU Interrupt Status Register | 10-92 |
| 10.7.2.7 | AFEU Interrupt Mask Register | 10-94 |
| 10.7.2.8 | AFEU End of Message Register | 10-96 |
| 10.7.2.9 | AFEU Context | 10-96 |

# Contents

Paragraph Number | Title | Page Number

10.7.2.9.1 Writing AFEU Context 10-96
10.7.2.9.2 Reading AFEU Context 10-97
10.7.2.10 AFEU Key Registers 10-97
10.7.2.10.1 AFEU FIFOs 10-97
10.7.3 Cyclical Redundancy Check Unit (CRCU) 10-98
10.7.3.1 ICV Checking in CRCU 10-98
10.7.3.2 CRCU Mode Register 10-98
10.7.3.3 CRCU Key Size Register 10-99
10.7.3.4 CRCU Data Size Register 10-100
10.7.3.5 CRCU Reset Control Register 10-100
10.7.3.6 CRCU Control Register 10-101
10.7.3.7 CRCU Status Register 10-101
10.7.3.8 CRCU Interrupt Status Register 10-102
10.7.3.9 CRCU Interrupt Mask Register 10-104
10.7.3.10 CRCU ICV Size Register 10-106
10.7.3.11 CRCU End of Message Register 10-106
10.7.3.12 CRCU Context Register 10-106
10.7.3.13 CRCU Key Register 10-108
10.7.3.14 CRCU FIFO 10-108
10.7.4 Data Encryption Standard Execution Unit (DEU) 10-108
10.7.4.1 DEU Mode Register 10-109
10.7.4.2 DEU Key Size Register 10-110
10.7.4.3 DEU Data Size Register (DEUDSR) 10-110
10.7.4.4 DEU Reset Control Register 10-111
10.7.4.5 DEU Status Register 10-112
10.7.4.6 DEU Interrupt Status Register 10-113
10.7.4.7 DEU Interrupt Mask Register 10-115
10.7.4.8 DEU End of Message Register 10-116
10.7.4.9 DEU IV Register 10-117
10.7.4.10 DEU Key Registers 10-117
10.7.4.11 DEU FIFOs 10-117
10.7.5 Kasumi Execution Unit (KEU) 10-117
10.7.5.1 KEU Mode Register (KEUMR) 10-118
10.7.5.2 KEU Key Size Register (KEUKSR) 10-119
10.7.5.3 KEU Data Size Register (KEUDSR) 10-120
10.7.5.4 KEU Reset Control Register (KEURCR) 10-121
10.7.5.5 KEU Status Register (KEUSR) 10-122
10.7.5.6 KEU Interrupt Status Register (KEUISR) 10-123
10.7.5.7 KEU Interrupt Mask Register (KEUIMR) 10-125
10.7.5.8 KEU Data Out Register (f9 MAC) (KEUDOR) 10-127
10.7.5.9 KEU End of Message Register (KEUEMR) 10-127

# Contents

| Paragraph Number | Title | Page Number |
|---|---|---|
| 10.7.5.10 | KEU IV_1 Register (KEUIV1) | 10-128 |
| 10.7.5.11 | KEU ICV_In Register (KEUICV) | 10-129 |
| 10.7.5.12 | KEU IV_2 Register (FRESH) (KEUIV2) | 10-129 |
| 10.7.5.13 | KEU Context Data Registers (KEUC*n*) | 10-129 |
| 10.7.5.14 | KEU Key Data Registers_1 and _2 (Confidentiality Key) (KEUKD*n*) | 10-130 |
| 10.7.5.15 | KEU Key Data Registers _3 and _4 (Integrity Key) (KEUKD*n*) | 10-130 |
| 10.7.5.16 | KEU FIFOs | 10-131 |
| 10.7.6 | Message Digest Execution Unit (MDEU) | 10-132 |
| 10.7.6.1 | ICV Checking in MDEU | 10-132 |
| 10.7.6.2 | MDEU Mode Register | 10-132 |
| 10.7.6.3 | Recommended Settings for MDEU Mode Register | 10-135 |
| 10.7.6.4 | MDEU Key Size Register | 10-136 |
| 10.7.6.5 | MDEU Data Size Register | 10-137 |
| 10.7.6.6 | MDEU Reset Control Register | 10-137 |
| 10.7.6.7 | MDEU Status Register | 10-138 |
| 10.7.6.8 | MDEU Interrupt Status Register | 10-139 |
| 10.7.6.9 | MDEU Interrupt Mask Register | 10-141 |
| 10.7.6.10 | MDEU ICV Size Register | 10-142 |
| 10.7.6.11 | MDEU End of Message Register | 10-143 |
| 10.7.6.12 | MDEU Context Registers | 10-143 |
| 10.7.6.13 | MDEU Key Registers | 10-146 |
| 10.7.6.14 | MDEU FIFOs | 10-146 |
| 10.7.7 | Public Key Execution Units (PKEU) | 10-146 |
| 10.7.7.1 | PKEU Mode Register | 10-147 |
| 10.7.7.2 | PKEU Key Size Register | 10-147 |
| 10.7.7.3 | PKEU AB Size Register | 10-148 |
| 10.7.7.4 | PKEU Data Size Register | 10-149 |
| 10.7.7.5 | PKEU Reset Control Register | 10-149 |
| 10.7.7.6 | PKEU Status Register | 10-150 |
| 10.7.7.7 | PKEU Interrupt Status Register | 10-151 |
| 10.7.7.8 | PKEU Interrupt Mask Register | 10-153 |
| 10.7.7.9 | PKEU End of Message Register | 10-154 |
| 10.7.7.10 | PKEU Parameter Memories | 10-154 |
| 10.7.7.10.1 | PKEU Parameter Memory A | 10-154 |
| 10.7.7.10.2 | PKEU Parameter Memory B | 10-155 |
| 10.7.7.10.3 | PKEU Parameter Memory E | 10-155 |
| 10.7.7.10.4 | PKEU Parameter Memory N | 10-155 |
| 10.7.8 | Random Number Generator Unit (RNGU) | 10-155 |
| 10.7.8.1 | RNGU Mode Register | 10-156 |
| 10.7.8.2 | RNGU Data Size Register | 10-156 |
| 10.7.8.3 | RNGU Reset Control Register | 10-156 |

# Contents

Paragraph Number | Title | Page Number

10.7.8.4 RNGU Status Register .......... 10-157
10.7.8.5 RNGU Interrupt Status Register .......... 10-158
10.7.8.6 RNGU Interrupt Mask Register .......... 10-159
10.7.8.7 RNGU End of Message Register .......... 10-160
10.7.8.8 RNGU Entropy Registers .......... 10-161
10.7.8.9 RNGU FIFO .......... 10-161

## Chapter 11
## I$^2$C Interfaces

11.1 Introduction .......... 11-1
11.1.1 Overview .......... 11-2
11.1.2 Features .......... 11-2
11.1.3 Modes of Operation .......... 11-2
11.2 External Signal Descriptions .......... 11-3
11.2.1 Signal Overview .......... 11-3
11.2.2 Detailed Signal Descriptions .......... 11-3
11.3 Memory Map/Register Definition .......... 11-4
11.3.1 Register Descriptions .......... 11-5
11.3.1.1 I$^2$C Address Register (I2CADR) .......... 11-6
11.3.1.2 I$^2$C Frequency Divider Register (I2CFDR) .......... 11-6
11.3.1.3 I$^2$C Control Register (I2CCR) .......... 11-7
11.3.1.4 I$^2$C Status Register (I2CSR) .......... 11-9
11.3.1.5 I$^2$C Data Register (I2CDR) .......... 11-10
11.3.1.6 Digital Filter Sampling Rate Register (I2CDFSRR) .......... 11-11
11.4 Functional Description .......... 11-11
11.4.1 Transaction Protocol .......... 11-11
11.4.1.1 START Condition .......... 11-12
11.4.1.2 Slave Address Transmission .......... 11-12
11.4.1.3 Repeated START Condition .......... 11-13
11.4.1.4 STOP Condition .......... 11-13
11.4.1.5 Protocol Implementation Details .......... 11-13
11.4.1.5.1 Transaction Monitoring—Implementation Details .......... 11-14
11.4.1.5.2 Control Transfer—Implementation Details .......... 11-14
11.4.1.6 Address Compare—Implementation Details .......... 11-15
11.4.2 Arbitration Procedure .......... 11-15
11.4.2.1 Arbitration Control .......... 11-15
11.4.3 Handshaking .......... 11-16
11.4.4 Clock Control .......... 11-16
11.4.4.1 Clock Synchronization .......... 11-16
11.4.4.2 Input Synchronization and Digital Filter .......... 11-16

# Contents

Paragraph Number | Title | Page Number

11.4.4.2.1 Input Signal Synchronization ... 11-16
11.4.4.2.2 Filtering of SCL and SDA Lines ... 11-17
11.4.4.3 Clock Stretching ... 11-17
11.4.5 Boot Sequencer Mode ... 11-17
11.4.5.1 EEPROM Calling Address ... 11-18
11.4.5.2 EEPROM Data Format ... 11-19
11.5 Initialization/Application Information ... 11-21
11.5.1 Initialization Sequence ... 11-21
11.5.2 Generation of START ... 11-21
11.5.3 Post-Transfer Software Response ... 11-22
11.5.4 Generation of STOP ... 11-22
11.5.5 Generation of Repeated START ... 11-23
11.5.6 Generation of SCL When SDA Low ... 11-23
11.5.7 Slave Mode Interrupt Service Routine ... 11-23
11.5.7.1 Slave Transmitter and Received Acknowledge ... 11-23
11.5.7.2 Loss of Arbitration and Forcing of Slave Mode ... 11-24
11.5.8 Interrupt Service Routine Flowchart ... 11-24

## Chapter 12 DUART

12.1 Overview ... 12-1
12.1.1 Features ... 12-1
12.1.2 Modes of Operation ... 12-2
12.2 External Signal Descriptions ... 12-3
12.3 Memory Map/Register Definition ... 12-3
12.3.1 Register Descriptions ... 12-5
12.3.1.1 Receiver Buffer Registers (URBR*n*) (ULCR[DLAB] = 0) ... 12-5
12.3.1.2 Transmitter Holding Registers (UTHR*n*) (ULCR[DLAB] = 0) ... 12-5
12.3.1.3 Divisor Most and Least Significant Byte Registers (UDMB and UDLB) (ULCR[DLAB] = 1) ... 12-6
12.3.1.4 Interrupt Enable Register (UIER) (ULCR[DLAB] = 0) ... 12-7
12.3.1.5 Interrupt ID Registers (UIIR*n*) (ULCR[DLAB] = 0) ... 12-8
12.3.1.6 FIFO Control Registers (UFCR*n*) (ULCR[DLAB] = 0) ... 12-10
12.3.1.7 Alternate Function Registers (UAFR*n*) (ULCR[DLAB] = 1) ... 12-11
12.3.1.8 Line Control Registers (ULCR*n*) ... 12-11
12.3.1.9 Modem Control Registers (UMCR*n*) ... 12-14
12.3.1.10 Line Status Registers (ULSR*n*) ... 12-15
12.3.1.11 Modem Status Registers (UMSR*n*) ... 12-16
12.3.1.12 Scratch Registers (USCR*n*) ... 12-17
12.3.1.13 DMA Status Registers (UDSR*n*) ... 12-17

# Contents

| Paragraph Number | Title | Page Number |
|---|---|---|
| 12.4 | Functional Description | 12-19 |
| 12.4.1 | Serial Interface | 12-19 |
| 12.4.1.1 | START Bit | 12-20 |
| 12.4.1.2 | Data Transfer | 12-20 |
| 12.4.1.3 | Parity Bit | 12-20 |
| 12.4.1.4 | STOP Bit | 12-20 |
| 12.4.2 | Baud-Rate Generator Logic | 12-20 |
| 12.4.3 | Local Loopback Mode | 12-21 |
| 12.4.4 | Errors | 12-21 |
| 12.4.4.1 | Framing Error | 12-21 |
| 12.4.4.2 | Parity Error | 12-22 |
| 12.4.4.3 | Overrun Error | 12-22 |
| 12.4.5 | FIFO Mode | 12-22 |
| 12.4.5.1 | FIFO Interrupts | 12-22 |
| 12.4.5.2 | DMA Mode Select | 12-22 |
| 12.4.5.3 | Interrupt Control Logic | 12-23 |
| 12.5 | DUART Initialization/Application Information | 12-23 |

## Chapter 13
## Enhanced Local Bus Controller

| Paragraph Number | Title | Page Number |
|---|---|---|
| 13.1 | Introduction | 13-1 |
| 13.1.1 | Overview | 13-2 |
| 13.1.2 | Features | 13-2 |
| 13.1.3 | Modes of Operation | 13-3 |
| 13.1.3.1 | eLBC Bus Clock and Clock Ratios | 13-4 |
| 13.1.3.2 | Source ID Debug Mode | 13-4 |
| 13.2 | External Signal Descriptions | 13-4 |
| 13.3 | Memory Map/Register Definition | 13-9 |
| 13.3.1 | Register Descriptions | 13-10 |
| 13.3.1.1 | Base Registers (BR0–BR7) | 13-11 |
| 13.3.1.2 | Option Registers (OR0–OR7) | 13-12 |
| 13.3.1.2.1 | Address Mask | 13-13 |
| 13.3.1.2.2 | Option Registers (OR*n*)—GPCM Mode | 13-14 |
| 13.3.1.2.3 | Option Registers (OR*n*)—FCM Mode | 13-16 |
| 13.3.1.2.4 | Option Registers (OR*n*)—UPM Mode | 13-19 |
| 13.3.1.3 | UPM Memory Address Register (MAR) | 13-20 |
| 13.3.1.4 | UPM Mode Registers (M*x*MR) | 13-21 |
| 13.3.1.5 | Memory Refresh Timer Prescaler Register (MRTPR) | 13-23 |
| 13.3.1.6 | UPM/FCM Data Register (MDR) | 13-23 |
| 13.3.1.7 | Special Operation Initiation Register (LSOR) | 13-24 |

# Contents

| Paragraph Number | Title | Page Number |
|---|---|---|
| 13.3.1.8 | UPM Refresh Timer (LURT) | 13-25 |
| 13.3.1.9 | Transfer Error Status Register (LTESR) | 13-26 |
| 13.3.1.10 | Transfer Error Check Disable Register (LTEDR) | 13-28 |
| 13.3.1.11 | Transfer Error Interrupt Enable Register (LTEIR) | 13-29 |
| 13.3.1.12 | Transfer Error Attributes Register (LTEATR) | 13-30 |
| 13.3.1.13 | Transfer Error Address Register (LTEAR) | 13-31 |
| 13.3.1.14 | Transfer Error ECC Register (LTECCR) | 13-31 |
| 13.3.1.15 | Local Bus Configuration Register (LBCR) | 13-32 |
| 13.3.1.16 | Clock Ratio Register (LCRR) | 13-34 |
| 13.3.1.17 | Flash Mode Register (FMR) | 13-35 |
| 13.3.1.18 | Flash Instruction Register (FIR) | 13-37 |
| 13.3.1.19 | Flash Command Register (FCR) | 13-38 |
| 13.3.1.20 | Flash Block Address Register (FBAR) | 13-39 |
| 13.3.1.21 | Flash Page Address Register (FPAR) | 13-39 |
| 13.3.1.22 | Flash Byte Count Register (FBCR) | 13-41 |
| 13.3.1.23 | Flash ECC Block*n* Register (FECC0–FECC3) | 13-41 |
| 13.4 | Functional Description | 13-42 |
| 13.4.1 | Basic Architecture | 13-43 |
| 13.4.1.1 | Address and Address Space Checking | 13-43 |
| 13.4.1.2 | External Address Latch Enable Signal (LALE) | 13-43 |
| 13.4.1.3 | Data Transfer Acknowledge (TA) | 13-45 |
| 13.4.1.4 | Data Buffer Control (LBCTL) | 13-46 |
| 13.4.1.5 | Atomic Operation | 13-46 |
| 13.4.1.6 | Parity Generation and Checking (LDP) | 13-47 |
| 13.4.1.7 | Bus Monitor | 13-47 |
| 13.4.1.8 | PLL Bypass Mode | 13-47 |
| 13.4.2 | General-Purpose Chip-Select Machine (GPCM) | 13-48 |
| 13.4.2.1 | GPCM Read Signal Timing | 13-49 |
| 13.4.2.2 | GPCM Write Signal Timing | 13-51 |
| 13.4.2.3 | Chip-Select Assertion Timing | 13-52 |
| 13.4.2.3.1 | Programmable Wait State Configuration | 13-53 |
| 13.4.2.3.2 | Chip-Select and Write Enable Negation Timing | 13-53 |
| 13.4.2.3.3 | Relaxed Timing | 13-54 |
| 13.4.2.3.4 | Output Enable (LOE) Timing | 13-56 |
| 13.4.2.3.5 | Extended Hold Time on Read Accesses | 13-56 |
| 13.4.2.4 | External Access Termination (LGTA) | 13-57 |
| 13.4.2.5 | GPCM Boot Chip-Select Operation | 13-58 |
| 13.4.3 | Flash Control Machine (FCM) | 13-59 |
| 13.4.3.1 | FCM Buffer RAM | 13-61 |
| 13.4.3.1.1 | Buffer Layout and Page Mapping for Small-Page NAND Flash Devices | 13-61 |
| 13.4.3.1.2 | Buffer Layout and Page Mapping for Large-Page NAND Flash Devices | 13-62 |

# Contents

| Paragraph Number | Title | Page Number |
|---|---|---|
| 13.4.3.1.3 | Error Correcting Codes and the Spare Region | 13-63 |
| 13.4.3.2 | Programming FCM | 13-65 |
| 13.4.3.2.1 | FCM Command Instructions | 13-66 |
| 13.4.3.2.2 | FCM No-Operation Instruction | 13-66 |
| 13.4.3.2.3 | FCM Address Instructions | 13-66 |
| 13.4.3.2.4 | FCM Data Read Instructions | 13-67 |
| 13.4.3.2.5 | FCM Data Write Instructions | 13-67 |
| 13.4.3.3 | FCM Signal Timing | 13-68 |
| 13.4.3.3.1 | FCM Chip-Select Timing | 13-68 |
| 13.4.3.3.2 | FCM Command, Address, and Write Data Timing | 13-68 |
| 13.4.3.3.3 | FCM Ready/Busy Timing | 13-70 |
| 13.4.3.3.4 | FCM Read Data Timing | 13-70 |
| 13.4.3.3.5 | FCM Extended Read Hold Timing | 13-71 |
| 13.4.3.4 | FCM Boot Chip-Select Operation | 13-72 |
| 13.4.3.4.1 | FCM Bank 0 Reset Initialization | 13-72 |
| 13.4.3.4.2 | Boot Block Loading into the FCM Buffer RAM | 13-73 |
| 13.4.4 | User-Programmable Machines (UPMs) | 13-74 |
| 13.4.4.1 | UPM Requests | 13-75 |
| 13.4.4.1.1 | Memory Access Requests | 13-76 |
| 13.4.4.1.2 | UPM Refresh Timer Requests | 13-77 |
| 13.4.4.1.3 | Software Requests—RUN Command | 13-77 |
| 13.4.4.1.4 | Exception Requests | 13-77 |
| 13.4.4.2 | Programming the UPMs | 13-78 |
| 13.4.4.2.1 | UPM Programming Example (Two Sequential Writes to the RAM Array) | 13-79 |
| 13.4.4.2.2 | UPM Programming Example (Two Sequential Reads from the RAM Array) | 13-79 |
| 13.4.4.3 | UPM Signal Timing | 13-80 |
| 13.4.4.4 | RAM Array | 13-80 |
| 13.4.4.4.1 | RAM Words | 13-80 |
| 13.4.4.4.2 | Chip-Select Signal Timing (CST*n*) | 13-84 |
| 13.4.4.4.3 | Byte Select Signal Timing (BST*n*) | 13-84 |
| 13.4.4.4.4 | General-Purpose Signals (G*n*T*n*, GO*n*) | 13-85 |
| 13.4.4.4.5 | Loop Control (LOOP) | 13-85 |
| 13.4.4.4.6 | Repeat Execution of Current RAM Word (REDO) | 13-86 |
| 13.4.4.4.7 | Address Multiplexing (AMX) | 13-86 |
| 13.4.4.4.8 | Data Valid and Data Sample Control (UTA) | 13-87 |
| 13.4.4.4.9 | LGPL[0:5] Signal Negation (LAST) | 13-88 |
| 13.4.4.4.10 | Wait Mechanism (WAEN) | 13-88 |
| 13.4.4.5 | Synchronous Sampling of LUPWAIT for Early Transfer Acknowledge | 13-89 |
| 13.4.4.6 | Extended Hold Time on Read Accesses | 13-90 |
| 13.5 | Initialization/Application Information | 13-90 |
| 13.5.1 | Interfacing to Peripherals in Different Address Modes | 13-90 |

# Contents

Paragraph Number | Title | Page Number

13.5.1.1 Multiplexed Address/Data Bus for 32-Bit Addressing ........ 13-90
13.5.1.2 Peripheral Hierarchy on the Local Bus for High Bus Speeds ........ 13-91
13.5.1.3 GPCM Timings ........ 13-92
13.5.2 Bus Turnaround ........ 13-92
13.5.2.1 Address Phase after Previous Read ........ 13-93
13.5.2.2 Read Data Phase after Address Phase ........ 13-93
13.5.2.3 Read-Modify-Write Cycle for Parity Protected Memory Banks ........ 13-93
13.5.2.4 UPM Cycles with Additional Address Phases ........ 13-93
13.5.3 Interface to Different Port-Size Devices ........ 13-93
13.5.4 Command Sequence Examples for NAND Flash EEPROM ........ 13-95
13.5.4.1 NAND Flash Soft Reset Command Sequence Example ........ 13-95
13.5.4.2 NAND Flash Read Status Command Sequence Example ........ 13-96
13.5.4.3 NAND Flash Read Identification Command Sequence Example ........ 13-96
13.5.4.4 NAND Flash Page Read Command Sequence Example ........ 13-97
13.5.4.5 NAND Flash Block Erase Command Sequence Example ........ 13-97
13.5.4.6 NAND Flash Program Command Sequence Example ........ 13-98
13.5.5 Interfacing to Fast-Page Mode DRAM Using UPM ........ 13-99
13.5.6 Interfacing to ZBT SRAM Using UPM ........ 13-104

## Chapter 14
## Enhanced Three-Speed Ethernet Controllers

14.1 Overview ........ 14-1
14.2 Features ........ 14-2
14.3 Modes of Operation ........ 14-5
14.4 External Signals Description ........ 14-6
14.4.1 Detailed Signal Descriptions ........ 14-9
14.5 Memory Map/Register Definition ........ 14-14
14.5.1 Top-Level Module Memory Map ........ 14-15
14.5.2 Detailed Memory Map ........ 14-15
14.5.3 Memory-Mapped Register Descriptions ........ 14-26
14.5.3.1 eTSEC General Control and Status Registers ........ 14-26
14.5.3.1.1 Controller ID Register (TSEC_ID) ........ 14-26
14.5.3.1.2 Controller ID Register (TSEC_ID2) ........ 14-27
14.5.3.1.3 Interrupt Event Register (IEVENT) ........ 14-27
14.5.3.1.4 Interrupt Mask Register (IMASK) ........ 14-31
14.5.3.1.5 Error Disabled Register (EDIS) ........ 14-33
14.5.3.1.6 Ethernet Control Register (ECNTRL) ........ 14-35
14.5.3.1.7 Pause Time Value Register (PTV) ........ 14-37
14.5.3.1.8 DMA Control Register (DMACTRL) ........ 14-38
14.5.3.1.9 TBI Physical Address Register (TBIPA) ........ 14-40

# Contents

| Paragraph Number | Title | Page Number |
|---|---|---|
| 14.5.3.2 | eTSEC Transmit Control and Status Registers | 14-40 |
| 14.5.3.2.1 | Transmit Control Register (TCTRL) | 14-40 |
| 14.5.3.2.2 | Transmit Status Register (TSTAT) | 14-42 |
| 14.5.3.2.3 | Default VLAN Control Word Register (DFVLAN) | 14-46 |
| 14.5.3.2.4 | Transmit Interrupt Coalescing Register (TXIC) | 14-47 |
| 14.5.3.2.5 | Transmit Queue Control Register (TQUEUE) | 14-48 |
| 14.5.3.2.6 | TxBD Ring 0–3 Weighting Register (TR03WT) | 14-49 |
| 14.5.3.2.7 | TxBD Ring 4–7 Weighting Register (TR47WT) | 14-49 |
| 14.5.3.2.8 | Transmit Data Buffer Pointer High Register (TBDBPH) | 14-50 |
| 14.5.3.2.9 | Transmit Buffer Descriptor Pointers 0–7 (TBPTR0–TBPTR7) | 14-50 |
| 14.5.3.2.10 | Transmit Descriptor Base Address High Register (TBASEH) | 14-51 |
| 14.5.3.2.11 | Transmit Descriptor Base Address Registers (TBASE0–TBASE7) | 14-52 |
| 14.5.3.2.12 | Transmit Time Stamp Identification Register (TMR_TXTS1–2_ID) | 14-52 |
| 14.5.3.2.13 | Transmit Time Stamp Register (TMR_TXTS1–2_H/L) | 14-53 |
| 14.5.3.3 | eTSEC Receive Control and Status Registers | 14-53 |
| 14.5.3.3.1 | Receive Control Register (RCTRL) | 14-54 |
| 14.5.3.3.2 | Receive Status Register (RSTAT) | 14-56 |
| 14.5.3.3.3 | Receive Interrupt Coalescing Register (RXIC) | 14-59 |
| 14.5.3.3.4 | Receive Queue Control Register (RQUEUE) | 14-60 |
| 14.5.3.3.5 | Receive Bit Field Extract Control Register (RBIFX) | 14-61 |
| 14.5.3.3.6 | Receive Queue Filer Table Address Register (RQFAR) | 14-63 |
| 14.5.3.3.7 | Receive Queue Filer Table Control Register (RQFCR) | 14-63 |
| 14.5.3.3.8 | Receive Queue Filer Table Property Register (RQFPR) | 14-65 |
| 14.5.3.3.9 | Maximum Receive Buffer Length Register (MRBLR) | 14-68 |
| 14.5.3.3.10 | Receive Data Buffer Pointer High Register (RBDBPH) | 14-68 |
| 14.5.3.3.11 | Receive Buffer Descriptor Pointers 0–7 (RBPTR0–RBPTR7) | 14-69 |
| 14.5.3.3.12 | Receive Descriptor Base Address High Register (RBASEH) | 14-70 |
| 14.5.3.3.13 | Receive Descriptor Base Address Registers (RBASE0–RBASE7) | 14-70 |
| 14.5.3.3.14 | Receive Stamp Register (TMR_RXTS_H/L) | 14-71 |
| 14.5.3.4 | MAC Functionality | 14-71 |
| 14.5.3.4.1 | Configuring the MAC | 14-71 |
| 14.5.3.4.2 | Controlling CSMA/CD | 14-71 |
| 14.5.3.4.3 | Handling Packet Collisions | 14-72 |
| 14.5.3.4.4 | Controlling Packet Flow | 14-72 |
| 14.5.3.4.5 | Controlling PHY Links | 14-73 |
| 14.5.3.5 | MAC Registers | 14-74 |
| 14.5.3.5.1 | MAC Configuration 1 Register (MACCFG1) | 14-74 |
| 14.5.3.5.2 | MAC Configuration 2 Register (MACCFG2) | 14-76 |
| 14.5.3.5.3 | Inter-Packet Gap/Inter-Frame Gap Register (IPGIFG) | 14-78 |
| 14.5.3.5.4 | Half-Duplex Register (HAFDUP) | 14-79 |
| 14.5.3.5.5 | Maximum Frame Length Register (MAXFRM) | 14-80 |

# Contents

Paragraph Number | Title | Page Number

14.5.3.5.6 MII Management Configuration Register (MIIMCFG) .......... 14-80
14.5.3.5.7 MII Management Command Register (MIIMCOM).......... 14-81
14.5.3.5.8 MII Management Address Register (MIIMADD).......... 14-82
14.5.3.5.9 MII Management Control Register (MIIMCON).......... 14-82
14.5.3.5.10 MII Management Status Register (MIIMSTAT) .......... 14-83
14.5.3.5.11 MII Management Indicator Register (MIIMIND).......... 14-83
14.5.3.5.12 Interface Status Register (IFSTAT).......... 14-84
14.5.3.5.13 MAC Station Address Part 1 Register (MACSTNADDR1) .......... 14-84
14.5.3.5.14 MAC Station Address Part 2 Register (MACSTNADDR2) .......... 14-85
14.5.3.5.15 MAC Exact Match Address 1–15 Part 1 Registers (MAC01ADDR1–MAC15ADDR1).......... 14-86
14.5.3.5.16 MAC Exact Match Address 1–15 Part 2 Registers (MAC01ADDR2–MAC15ADDR2).......... 14-86
14.5.3.6 MIB Registers.......... 14-87
14.5.3.6.1 Transmit and Receive 64-Byte Frame Counter (TR64) .......... 14-88
14.5.3.6.2 Transmit and Receive 65- to 127-Byte Frame Counter (TR127).......... 14-88
14.5.3.6.3 Transmit and Receive 128- to 255-Byte Frame Counter (TR255).......... 14-89
14.5.3.6.4 Transmit and Receive 256- to 511-Byte Frame Counter (TR511) .......... 14-89
14.5.3.6.5 Transmit and Receive 512- to 1023-Byte Frame Counter (TR1K) .......... 14-90
14.5.3.6.6 Transmit and Receive 1024- to 1518-Byte Frame Counter (TRMAX).......... 14-90
14.5.3.6.7 Transmit and Receive 1519- to 1522-Byte VLAN Frame Counter (TRMGV)......... 14-91
14.5.3.6.8 Receive Byte Counter (RBYT).......... 14-91
14.5.3.6.9 Receive Packet Counter (RPKT).......... 14-92
14.5.3.6.10 Receive FCS Error Counter (RFCS) .......... 14-92
14.5.3.6.11 Receive Multicast Packet Counter (RMCA) .......... 14-93
14.5.3.6.12 Receive Broadcast Packet Counter (RBCA) .......... 14-93
14.5.3.6.13 Receive Control Frame Packet Counter (RXCF) .......... 14-94
14.5.3.6.14 Receive Pause Frame Packet Counter (RXPF).......... 14-94
14.5.3.6.15 Receive Unknown Opcode Packet Counter (RXUO).......... 14-95
14.5.3.6.16 Receive Alignment Error Counter (RALN) .......... 14-95
14.5.3.6.17 Receive Frame Length Error Counter (RFLR).......... 14-96
14.5.3.6.18 Receive Code Error Counter (RCDE) .......... 14-96
14.5.3.6.19 Receive Carrier Sense Error Counter (RCSE).......... 14-97
14.5.3.6.20 Receive Undersize Packet Counter (RUND).......... 14-97
14.5.3.6.21 Receive Oversize Packet Counter (ROVR).......... 14-98
14.5.3.6.22 Receive Fragments Counter (RFRG) .......... 14-98
14.5.3.6.23 Receive Jabber Counter (RJBR).......... 14-99
14.5.3.6.24 Receive Dropped Packet Counter (RDRP).......... 14-99
14.5.3.6.25 Transmit Byte Counter (TBYT) .......... 14-100
14.5.3.6.26 Transmit Packet Counter (TPKT).......... 14-100

# Contents

| Paragraph Number | Title | Page Number |
|---|---|---|
| 14.5.3.6.27 | Transmit Multicast Packet Counter (TMCA) | 14-101 |
| 14.5.3.6.28 | Transmit Broadcast Packet Counter (TBCA) | 14-101 |
| 14.5.3.6.29 | Transmit Pause Control Frame Counter (TXPF) | 14-102 |
| 14.5.3.6.30 | Transmit Deferral Packet Counter (TDFR) | 14-102 |
| 14.5.3.6.31 | Transmit Excessive Deferral Packet Counter (TEDF) | 14-103 |
| 14.5.3.6.32 | Transmit Single Collision Packet Counter (TSCL) | 14-103 |
| 14.5.3.6.33 | Transmit Multiple Collision Packet Counter (TMCL) | 14-104 |
| 14.5.3.6.34 | Transmit Late Collision Packet Counter (TLCL) | 14-104 |
| 14.5.3.6.35 | Transmit Excessive Collision Packet Counter (TXCL) | 14-105 |
| 14.5.3.6.36 | Transmit Total Collision Counter (TNCL) | 14-105 |
| 14.5.3.6.37 | Transmit Drop Frame Counter (TDRP) | 14-106 |
| 14.5.3.6.38 | Transmit Jabber Frame Counter (TJBR) | 14-106 |
| 14.5.3.6.39 | Transmit FCS Error Counter (TFCS) | 14-107 |
| 14.5.3.6.40 | Transmit Control Frame Counter (TXCF) | 14-107 |
| 14.5.3.6.41 | Transmit Oversize Frame Counter (TOVR) | 14-108 |
| 14.5.3.6.42 | Transmit Undersize Frame Counter (TUND) | 14-108 |
| 14.5.3.6.43 | Transmit Fragment Counter (TFRG) | 14-109 |
| 14.5.3.6.44 | Carry Register 1 (CAR1) | 14-109 |
| 14.5.3.6.45 | Carry Register 2 (CAR2) | 14-111 |
| 14.5.3.6.46 | Carry Mask Register 1 (CAM1) | 14-112 |
| 14.5.3.6.47 | Carry Mask Register 2 (CAM2) | 14-113 |
| 14.5.3.6.48 | Receive Filer Rejected Packet Counter (RREJ) | 14-114 |
| 14.5.3.7 | Hash Function Registers | 14-115 |
| 14.5.3.7.1 | Individual/Group Address Registers 0–7 (IGADDR*n*) | 14-115 |
| 14.5.3.7.2 | Group Address Registers 0–7 (GADDR*n*) | 14-116 |
| 14.5.3.8 | FIFO Registers | 14-116 |
| 14.5.3.8.1 | FIFO Configuration Register (FIFOCFG) | 14-116 |
| 14.5.3.9 | DMA Attribute Registers | 14-118 |
| 14.5.3.9.1 | Attribute Register (ATTR) | 14-118 |
| 14.5.3.9.2 | Attribute Extract Length and Extract Index Register (ATTRELI) | 14-119 |
| 14.5.3.10 | Lossless Flow Control Configuration Registers | 14-120 |
| 14.5.3.10.1 | Receive Queue Parameters 0–7 (RQPRM0–PQPRM7) | 14-120 |
| 14.5.3.10.2 | Receive Free Buffer Descriptor Pointer Registers 0–7 (RFBPTR0–RFBPTR7) | 14-121 |
| 14.5.3.11 | Hardware Assist for IEEE1588 Compliant Timestamping | 14-122 |
| 14.5.3.11.1 | Timer Control Register (TMR_CTRL) | 14-122 |
| 14.5.3.11.2 | Timer Event Register (TMR_TEVENT) | 14-124 |
| 14.5.3.11.3 | Timer Event Mask Register (TMR_TEMASK) | 14-125 |
| 14.5.3.11.4 | Timer PTP Packet Event Register (TMR_PEVENT) | 14-126 |
| 14.5.3.11.5 | Timer Event Mask Register (TMR_PEMASK) | 14-127 |
| 14.5.3.11.6 | Timer Status Register (TMR_STAT) | 14-128 |

# Contents

Paragraph Number — Title — Page Number

14.5.3.11.7 Timer Counter Register (TMR_CNT_H/L) ... 14-128
14.5.3.11.8 Timer Drift Compensation Addend Register (TMR_ADD) ... 14-129
14.5.3.11.9 Timer Accumulator Register (TMR_ACC) ... 14-130
14.5.3.11.10 Timer Prescale Register (TMR_PRSC) ... 14-130
14.5.3.11.11 Timer Offset Register (TMROFF_H/L) ... 14-131
14.5.3.11.12 Alarm Time Comparator Register (TMR_ALARM1–2_H/L) ... 14-131
14.5.3.11.13 Timer Fixed Interval Period Register (TMR_FIPER1–3) ... 14-132
14.5.3.11.14 External Trigger Stamp Register (TMR_ETTS1–2_H/L) ... 14-133
14.5.4 Ten-Bit Interface (TBI) ... 14-133
14.5.4.1 TBI Transmit Process ... 14-134
14.5.4.1.1 Packet Encapsulation ... 14-134
14.5.4.1.2 8B10B Encoding ... 14-134
14.5.4.1.3 Preamble Shortening ... 14-134
14.5.4.2 TBI Receive Process ... 14-134
14.5.4.2.1 Synchronization ... 14-134
14.5.4.2.2 Auto-Negotiation for 1000BASE-X ... 14-135
14.5.4.3 TBI MII Set Register Descriptions ... 14-135
14.5.4.3.1 Control Register (CR) ... 14-136
14.5.4.3.2 Status Register (SR) ... 14-137
14.5.4.3.3 AN Advertisement Register (ANA) ... 14-138
14.5.4.3.4 AN Link Partner Base Page Ability Register (ANLPBPA) ... 14-140
14.5.4.3.5 AN Expansion Register (ANEX) ... 14-141
14.5.4.3.6 AN Next Page Transmit Register (ANNPT) ... 14-141
14.5.4.3.7 AN Link Partner Ability Next Page Register (ANLPANP) ... 14-142
14.5.4.3.8 Extended Status Register (EXST) ... 14-143
14.5.4.3.9 Jitter Diagnostics Register (JD) ... 14-144
14.5.4.3.10 TBI Control Register (TBICON) ... 14-145
14.6 Functional Description ... 14-146
14.6.1 Connecting to Physical Interfaces on Ethernet ... 14-146
14.6.1.1 Media-Independent Interface (MII) ... 14-146
14.6.1.2 Reduced Media-Independent Interface (RMII) ... 14-146
14.6.1.3 Gigabit Media-Independent Interface (GMII) ... 14-148
14.6.1.4 Reduced Gigabit Media-Independent Interface (RGMII) ... 14-148
14.6.1.5 Ten-Bit Interface (TBI) ... 14-149
14.6.1.6 Reduced Ten-Bit Interface (RTBI) ... 14-150
14.6.1.7 Ethernet Physical Interfaces Signal Summary ... 14-152
14.6.1.8 SGMII Interface ... 14-156
14.6.2 Connecting to FIFO Interfaces ... 14-156
14.6.2.1 Flow Control ... 14-157
14.6.2.2 CRC Appending and Checking ... 14-157
14.6.2.3 8-Bit GMII-Style Packet FIFO Mode ... 14-158

# Contents

Paragraph Number | Title | Page Number

14.6.2.4 8-Bit Encoded Packet FIFO Mode ........ 14-159
14.6.2.5 FIFO Interface Signal Summary ........ 14-159
14.6.3 Gigabit Ethernet Controller Channel Operation ........ 14-160
14.6.3.1 Initialization Sequence ........ 14-160
14.6.3.1.1 Hardware Controlled Initialization ........ 14-160
14.6.3.1.2 User Initialization ........ 14-160
14.6.3.2 Soft Reset and Reconfiguring Procedure ........ 14-161
14.6.3.3 Gigabit Ethernet Frame Transmission ........ 14-162
14.6.3.4 Gigabit Ethernet Frame Reception ........ 14-163
14.6.3.5 Ethernet Preamble Customization ........ 14-164
14.6.3.5.1 User-Defined Preamble Transmission ........ 14-165
14.6.3.5.2 User-Visible Preamble Reception ........ 14-165
14.6.3.6 RMON Support ........ 14-166
14.6.3.7 Frame Recognition ........ 14-166
14.6.3.7.1 Destination Address Recognition and Frame Filtering ........ 14-167
14.6.3.7.2 Hash Table Algorithm ........ 14-168
14.6.3.8 Magic Packet Mode ........ 14-170
14.6.3.9 Flow Control ........ 14-170
14.6.3.10 Interrupt Handling ........ 14-171
14.6.3.10.1 Interrupt Coalescing ........ 14-172
14.6.3.10.2 Interrupt Coalescing By Frame Count Threshold ........ 14-172
14.6.3.10.3 Interrupt Coalescing By Timer Threshold ........ 14-173
14.6.3.11 Inter-Frame Gap Time ........ 14-174
14.6.3.12 Internal and External Loop Back ........ 14-175
14.6.3.13 Error-Handling Procedure ........ 14-175
14.6.4 TCP/IP Off-Load ........ 14-176
14.6.4.1 Frame Control Blocks ........ 14-177
14.6.4.2 Transmit Path Off-Load and Tx PTP Packet Parsing ........ 14-178
14.6.4.3 Receive Path Off-Load ........ 14-179
14.6.5 Quality of Service (QoS) Provision ........ 14-181
14.6.5.1 Receive Parser ........ 14-181
14.6.5.2 Receive Queue Filer ........ 14-183
14.6.5.2.1 Filing Rules ........ 14-184
14.6.5.2.2 Comparing Properties with Bit Masks ........ 14-185
14.6.5.2.3 Special-Case Rules ........ 14-186
14.6.5.2.4 Filer Interrupt Events ........ 14-186
14.6.5.2.5 Setting Up the Receive Queue Filer Table ........ 14-187
14.6.5.2.6 Filer Example—802.1p Priority Filing ........ 14-187
14.6.5.2.7 Filer Example—IP Diff-Serv Code Points Filing ........ 14-188
14.6.5.2.8 Filer Example—TCP and UDP Port Filing ........ 14-188
14.6.5.3 Transmission Scheduling ........ 14-189

# Contents

Paragraph Number | Title | Page Number

14.6.5.3.1 Priority-Based Queuing (PBQ) 14-190
14.6.5.3.2 Modified Weighted Round-Robin Queuing (MWRR) 14-190
14.6.6 Lossless Flow Control 14-191
14.6.6.1 Back Pressure Determination through Free Buffers 14-191
14.6.6.2 Software Use of Hardware-Initiated Back Pressure 14-193
14.6.6.2.1 Initialization 14-193
14.6.6.2.2 Operation 14-193
14.6.7 Hardware Assist for IEEE Std. 1588-CompatibleTimestamping 14-194
14.6.7.1 Features 14-194
14.6.7.2 Timer Logic Overview 14-195
14.6.7.3 Time-Stamp Insertion on the Received Packets 14-195
14.6.7.3.1 Timestamp Point 14-195
14.6.7.4 PTP Packet Parsing 14-196
14.6.7.4.1 General Purpose Filer Rule 14-197
14.6.7.5 Time-Stamp Insertion on Transmit Packets 14-197
14.6.7.5.1 Interrupts 14-197
14.6.7.5.2 Error Condition 14-199
14.6.7.6 Tx PTP Packet Parsing 14-199
14.6.8 Buffer Descriptors 14-200
14.6.8.1 Data Buffer Descriptors 14-201
14.6.8.2 Transmit Data Buffer Descriptors (TxBD) 14-202
14.6.8.3 Receive Buffer Descriptors (RxBD) 14-205
14.7 Initialization/Application Information 14-207
14.7.1 Interface Mode Configuration 14-207
14.7.1.1 MII Interface Mode 14-208
14.7.1.2 GMII Interface Mode 14-212
14.7.1.3 TBI Interface Mode 14-216
14.7.1.4 RGMII Interface Mode 14-220
14.7.1.5 RMII Interface Mode 14-224
14.7.1.6 RTBI Interface Mode 14-228
14.7.1.7 8-Bit FIFO Mode 14-232
14.7.1.8 SGMII Interface Support 14-234

## Chapter 15
## DMA Controller

15.1 Introduction 15-1
15.1.1 Block Diagram 15-1
15.1.2 Overview 15-2
15.1.3 Features 15-2
15.1.4 Modes of Operation 15-2

# Contents

| Paragraph Number | Title | Page Number |
|---|---|---|
| 15.2 | External Signal Description | 15-4 |
| 15.2.1 | Signal Overview | 15-4 |
| 15.2.2 | Detailed Signal Descriptions | 15-5 |
| 15.3 | Memory Map/Register Definition | 15-5 |
| 15.3.1 | DMA Register Descriptions | 15-8 |
| 15.3.1.1 | Mode Registers (MR*n*) | 15-9 |
| 15.3.1.2 | Status Registers (SR*n*) | 15-11 |
| 15.3.1.3 | Current Link Descriptor Address Registers (CLNDAR*n* and ECLNDAR*n*) | 15-13 |
| 15.3.1.4 | Source Attributes Registers (SATR*n*) | 15-15 |
| 15.3.1.5 | Source Address Registers (SAR*n*) | 15-15 |
| 15.3.1.6 | Destination Attributes Registers (DATR*n*) | 15-16 |
| 15.3.1.7 | Destination Address Registers (DAR*n*) | 15-17 |
| 15.3.1.8 | Byte Count Registers (BCR*n*) | 15-18 |
| 15.3.1.9 | Next Link Descriptor Address Registers (NLNDAR*n* and ENLNDAR*n*) | 15-18 |
| 15.3.1.10 | Current List Descriptor Address Registers (CLSDAR*n* and ECLSDAR*n*) | 15-19 |
| 15.3.1.11 | Next List Descriptor Address Registers (NLSDAR*n* and ENLSDAR*n*) | 15-21 |
| 15.3.1.12 | Source Stride Registers (SSR*n*) | 15-22 |
| 15.3.1.13 | Destination Stride Registers (DSR*n*) | 15-22 |
| 15.3.1.14 | DMA General Status Register (DGSR) | 15-23 |
| 15.4 | Functional Description | 15-24 |
| 15.4.1 | DMA Channel Operation | 15-24 |
| 15.4.1.1 | Basic DMA Mode Transfer | 15-25 |
| 15.4.1.1.1 | Basic Direct Mode | 15-25 |
| 15.4.1.1.2 | Basic Direct Single-Write Start Mode | 15-26 |
| 15.4.1.1.3 | Basic Chaining Mode | 15-26 |
| 15.4.1.1.4 | Basic Chaining Single-Write Start Mode | 15-27 |
| 15.4.1.2 | Extended DMA Mode Transfer | 15-27 |
| 15.4.1.2.1 | Extended Direct Mode | 15-27 |
| 15.4.1.2.2 | Extended Direct Single-Write Start Mode | 15-28 |
| 15.4.1.2.3 | Extended Chaining Mode | 15-28 |
| 15.4.1.2.4 | Extended Chaining Single-Write Start Mode | 15-28 |
| 15.4.1.3 | External Control Mode Transfer | 15-29 |
| 15.4.1.4 | Channel Continue Mode for Cascading Transfer Chains | 15-30 |
| 15.4.1.4.1 | Basic Mode | 15-30 |
| 15.4.1.4.2 | Extended Mode | 15-31 |
| 15.4.1.5 | Channel Abort | 15-31 |
| 15.4.1.6 | Bandwidth Control | 15-31 |
| 15.4.1.7 | Channel State | 15-31 |
| 15.4.1.8 | Illustration of Stride Size and Stride Distance | 15-32 |
| 15.4.2 | DMA Transfer Interfaces | 15-32 |
| 15.4.3 | DMA Errors | 15-32 |

# Contents

Paragraph Number | Title | Page Number

15.4.4 DMA Descriptors ........ 15-33
15.4.5 Limitations and Restrictions ........ 15-36
15.5 DMA System Considerations ........ 15-38
15.5.1 Unusual DMA Scenarios ........ 15-39
15.5.1.1 DMA to  Core ........ 15-39
15.5.1.2 DMA to Configuration, Control, and Status Registers ........ 15-39
15.5.1.3 DMA to I$^2$C ........ 15-39
15.5.1.4 DMA to DUART ........ 15-39

## Chapter 16
## PCI Bus Interface

16.1 Introduction ........ 16-1
16.1.1 Overview ........ 16-2
16.1.1.1 Outbound Transactions ........ 16-3
16.1.1.2 Inbound Transactions ........ 16-4
16.1.2 Features ........ 16-4
16.1.3 Modes of Operation ........ 16-4
16.1.3.1 Host/Agent Mode Configuration ........ 16-5
16.1.3.1.1 Host Mode ........ 16-5
16.1.3.1.2 Agent Mode ........ 16-5
16.1.3.1.3 Agent Configuration Lock Mode ........ 16-5
16.1.3.2 PCI Clocking Configuration ........ 16-5
16.1.3.3 PCI Arbiter (Internal/External Arbiter) Configuration ........ 16-5
16.1.3.4 PCI Impedance Configuration ........ 16-5
16.2 External Signal Descriptions ........ 16-6
16.3 Memory Map/Register Definitions ........ 16-11
16.3.1 PCI Memory-Mapped Registers ........ 16-11
16.3.1.1 PCI Configuration Access Registers ........ 16-14
16.3.1.1.1 PCI Configuration Address Register (CFG_ADDR) ........ 16-14
16.3.1.1.2 PCI Configuration Data Register (CFG_DATA) ........ 16-15
16.3.1.1.3 PCI Interrupt Acknowledge Register (INT_ACK) ........ 16-15
16.3.1.2 PCI ATMU Outbound Registers ........ 16-15
16.3.1.2.1 PCI Outbound Translation Address Registers (POTAR*n*) ........ 16-16
16.3.1.2.2 PCI Outbound Translation Extended Address Registers (POTEAR*n*) ........ 16-16
16.3.1.2.3 PCI Outbound Window Base Address Registers (POWBAR*n*) ........ 16-17
16.3.1.2.4 PCI Outbound Window Attributes Registers (POWAR*n*) ........ 16-17
16.3.1.3 PCI ATMU Inbound Registers ........ 16-19
16.3.1.3.1 PCI Inbound Translation Address Registers (PITAR*n*) ........ 16-20
16.3.1.3.2 PCI Inbound Window Base Address Registers (PIWBAR*n*) ........ 16-20

# Contents

Paragraph Number — Title — Page Number

16.3.1.3.3 PCI Inbound Window Base Extended Address Registers (PIWBEAR*n*) ........ 16-21
16.3.1.3.4 PCI Inbound Window Attributes Registers (PIWAR*n*) ........ 16-21
16.3.1.4 PCI Error Management Registers ........ 16-23
16.3.1.4.1 PCI Error Detect Register (ERR_DR) ........ 16-24
16.3.1.4.2 PCI Error Capture Disable Register (ERR_CAP_DR) ........ 16-25
16.3.1.4.3 PCI Error Enable Register (ERR_EN) ........ 16-26
16.3.1.4.4 PCI Error Attributes Capture Register (ERR_ATTRIB) ........ 16-27
16.3.1.4.5 PCI Error Address Capture Register (ERR_ADDR) ........ 16-28
16.3.1.4.6 PCI Error Extended Address Capture Register (ERR_EXT_ADDR) ........ 16-28
16.3.1.4.7 PCI Error Data Low Capture Register (ERR_DL) ........ 16-29
16.3.1.4.8 PCI Error Data High Capture Register (ERR_DH) ........ 16-29
16.3.1.4.9 PCI Gasket Timer Register (GAS_TIMR) ........ 16-29
16.3.2 PCI Configuration Header ........ 16-30
16.3.2.1 PCI Vendor ID Register—Offset 0x00 ........ 16-30
16.3.2.2 PCI Device ID Register—Offset 0x02 ........ 16-31
16.3.2.3 PCI Bus Command Register—Offset 0x04 ........ 16-31
16.3.2.4 PCI Bus Status Register—Offset 0x06 ........ 16-32
16.3.2.5 PCI Revision ID Register—Offset 0x08 ........ 16-34
16.3.2.6 PCI Bus Programming Interface Register—Offset 0x09 ........ 16-34
16.3.2.7 PCI Subclass Code Register—Offset 0x0A ........ 16-35
16.3.2.8 PCI Bus Base Class Code Register—Offset 0x0B ........ 16-35
16.3.2.9 PCI Bus Cache Line Size Register—Offset 0x0C ........ 16-35
16.3.2.10 PCI Bus Latency Timer Register—0x0D ........ 16-36
16.3.2.11 PCI Base Address Registers ........ 16-36
16.3.2.12 PCI Subsystem Vendor ID Register ........ 16-38
16.3.2.13 PCI Subsystem ID Register ........ 16-39
16.3.2.14 PCI Bus Capabilities Pointer Register ........ 16-39
16.3.2.15 PCI Bus Interrupt Line Register ........ 16-39
16.3.2.16 PCI Bus Interrupt Pin Register ........ 16-40
16.3.2.17 PCI Bus Minimum Grant Register (MIN_GNT) ........ 16-40
16.3.2.18 PCI Bus Maximum Latency Register (MAX_LAT) ........ 16-41
16.3.2.19 PCI Bus Function Register (PBFR) ........ 16-41
16.3.2.20 PCI Bus Arbiter Configuration Register (PBACR) ........ 16-41
16.4 Functional Description ........ 16-42
16.4.1 PCI Bus Arbitration ........ 16-42
16.4.1.1 PCI Bus Arbiter Operation ........ 16-43
16.4.1.2 PCI Bus Parking ........ 16-44
16.4.1.3 Broken Master Lock-Out ........ 16-44
16.4.1.4 Power-Saving Modes and the PCI Arbiter ........ 16-45

# Contents

Paragraph Number | Title | Page Number

16.4.2 PCI Bus Protocol ........ 16-45
16.4.2.1 Basic Transfer Control ........ 16-45
16.4.2.2 PCI Bus Commands ........ 16-46
16.4.2.3 Addressing ........ 16-47
16.4.2.3.1 Memory Space Addressing ........ 16-47
16.4.2.3.2 I/O Space Addressing ........ 16-48
16.4.2.3.3 Configuration Space Addressing ........ 16-48
16.4.2.4 Device Selection ........ 16-48
16.4.2.5 Byte Alignment ........ 16-49
16.4.2.6 Bus Driving and Turnaround ........ 16-49
16.4.2.7 PCI Bus Transactions ........ 16-50
16.4.2.7.1 PCI Read Transactions ........ 16-50
16.4.2.7.2 PCI Write Transactions ........ 16-51
16.4.2.8 Transaction Termination ........ 16-52
16.4.2.8.1 Master-Initiated Termination ........ 16-52
16.4.2.8.2 Target-Initiated Termination ........ 16-53
16.4.2.9 Fast Back-to-Back Transactions ........ 16-56
16.4.2.10 Dual Address Cycles ........ 16-56
16.4.2.11 Configuration Cycles ........ 16-58
16.4.2.11.1 PCI Configuration Space Header ........ 16-58
16.4.2.11.2 Host Accessing the PCI Configuration Space ........ 16-60
16.4.2.11.3 Agent Accessing the PCI Configuration Space ........ 16-61
16.4.2.11.4 PCI Type 0 Configuration Translation ........ 16-62
16.4.2.11.5 Type 1 Configuration Translation ........ 16-63
16.4.2.12 Other Bus Transactions ........ 16-63
16.4.2.12.1 Interrupt-Acknowledge Transactions ........ 16-63
16.4.2.12.2 Special-Cycle Transactions ........ 16-64
16.4.2.13 PCI Error Functions ........ 16-65
16.4.2.13.1 PCI Parity ........ 16-65
16.4.2.13.2 Error Reporting ........ 16-66
16.5 Initialization/Application Information ........ 16-67
16.5.1 Power-On Reset Configuration Modes ........ 16-67
16.5.1.1 Host Mode ........ 16-68
16.5.1.2 Agent Mode ........ 16-68
16.5.1.3 Agent Configuration Lock Mode ........ 16-68
16.5.2 Byte Ordering ........ 16-69
16.5.2.1 Byte Order for Configuration Transactions ........ 16-70

Chapter 17
PCI Express Interface Controller

# Contents

**Paragraph Number** **Title** **Page Number**

17.1 Introduction ........ 17-1
17.1.1 Overview ........ 17-1
17.1.1.1 Outbound Transactions ........ 17-2
17.1.1.2 Inbound Transactions ........ 17-3
17.1.2 Features ........ 17-3
17.1.3 Modes of Operation ........ 17-4
17.1.3.1 Root Complex/Endpoint Modes ........ 17-4
17.1.3.2 Link Width ........ 17-4
17.2 External Signal Descriptions ........ 17-4
17.3 Memory Map/Register Definitions ........ 17-5
17.3.1 PCI Express Memory Mapped Registers ........ 17-6
17.3.2 PCI Express Configuration Access Registers ........ 17-10
17.3.2.1 PCI Express Configuration Address Register (PEX_CONFIG_ADDR) ........ 17-10
17.3.2.2 PCI Express Configuration Data Register (PEX_CONFIG_DATA) ........ 17-10
17.3.2.3 PCI Express Outbound Completion Timeout Register (PEX_OTB_CPL_TOR) ........ 17-11
17.3.2.4 PCI Express Configuration Retry Timeout Register (PEX_CONF_RTY_TOR) ........ 17-12
17.3.2.5 PCI Express Configuration Register (PEX_CONFIG) ........ 17-12
17.3.3 PCI Express Power Management Event and Message Registers ........ 17-13
17.3.3.1 PCI Express PME and Message Detect Register (PEX_PME_MES_DR) ........ 17-13
17.3.3.2 PCI Express PME and Message Disable Register (PEX_PME_MES_DISR) ........ 17-15
17.3.3.3 PCI Express PME and Message Interrupt Enable Register (PEX_PME_MES_IER) ........ 17-16
17.3.3.4 PCI Express Power Management Command Register (PEX_PMCR) ........ 17-18
17.3.4 PCI Express IP Block Revision Registers ........ 17-19
17.3.4.1 IP Block Revision Register 1 (PEX_IP_BLK_REV1) ........ 17-19
17.3.4.2 IP Block Revision Register 2 (PEX_IP_BLK_REV2) ........ 17-19
17.3.5 PCI Express ATMU Registers ........ 17-20
17.3.5.1 PCI Express Outbound ATMU Registers ........ 17-20
17.3.5.1.1 PCI Express Outbound Translation Address Registers (PEXOTAR*n*) ........ 17-20
17.3.5.1.2 PCI Express Outbound Translation Extended Address Registers (PEXOTEAR*n*) ........ 17-21
17.3.5.1.3 PCI Express Outbound Window Base Address Registers (PEXOWBAR*n*) ........ 17-22
17.3.5.1.4 PCI Express Outbound Window Attributes Registers (PEXOWAR*n*) ........ 17-22
17.3.5.2 PCI Express Inbound ATMU Registers ........ 17-25
17.3.5.2.1 EP Inbound ATMU Implementation ........ 17-25
17.3.5.2.2 RC Inbound ATMU Implementation ........ 17-25
17.3.5.2.3 PCI Express Inbound Translation Address Registers (PEXITAR*n*) ........ 17-26

# Contents

Paragraph Number | Title | Page Number

17.3.5.2.4 PCI Express Inbound Window Base Address Registers (PEXIWBAR*n*) ....... 17-27
17.3.5.2.5 PCI Express Inbound Window Base Extended Address Registers (PEXIWBEAR*n*) ....... 17-27
17.3.5.2.6 PCI Express Inbound Window Attributes Registers (PEXIWAR*n*) ....... 17-28
17.3.6 PCI Express Error Management Registers ....... 17-30
17.3.6.1 PCI Express Error Detect Register (PEX_ERR_DR) ....... 17-30
17.3.6.2 PCI Express Error Interrupt Enable Register (PEX_ERR_EN) ....... 17-32
17.3.6.3 PCI Express Error Disable Register (PEX_ERR_DISR) ....... 17-34
17.3.6.4 PCI Express Error Capture Status Register (PEX_ERR_CAP_STAT) ....... 17-36
17.3.6.5 PCI Express Error Capture Register 0 (PEX_ERR_CAP_R0) ....... 17-36
17.3.6.5.1 PEX_ERR_CAP_R0—Outbound Case ....... 17-37
17.3.6.5.2 PEX_ERR_CAP_R0—Inbound Case ....... 17-37
17.3.6.6 PCI Express Error Capture Register 1 (PEX_ERR_CAP_R1) ....... 17-38
17.3.6.6.1 PEX_ERR_CAP_R1—Outbound Case ....... 17-38
17.3.6.6.2 PEX_ERR_CAP_R1—Inbound Case ....... 17-39
17.3.6.7 PCI Express Error Capture Register 2 (PEX_ERR_CAP_R2) ....... 17-40
17.3.6.7.1 PEX_ERR_CAP_R2—Outbound Case ....... 17-40
17.3.6.7.2 PEX_ERR_CAP_R2—Inbound Case ....... 17-41
17.3.6.8 PCI Express Error Capture Register 3 (PEX_ERR_CAP_R3) ....... 17-42
17.3.6.8.1 PEX_ERR_CAP_R3—Outbound Case ....... 17-42
17.3.6.8.2 PEX_ERR_CAP_R3—Inbound Case ....... 17-42
17.3.7 PCI Express Configuration Space Access ....... 17-43
17.3.7.1 RC Configuration Register Access ....... 17-43
17.3.7.1.1 PCI Express Configuration Access Register Mechanism ....... 17-43
17.3.7.1.2 Outbound ATMU Configuration Mechanism (RC-Only) ....... 17-44
17.3.7.2 EP Configuration Register Access ....... 17-44
17.3.8 PCI Compatible Configuration Headers ....... 17-45
17.3.8.1 Common PCI Compatible Configuration Header Registers ....... 17-45
17.3.8.1.1 PCI Express Vendor ID Register—Offset 0x00 ....... 17-45
17.3.8.1.2 PCI Express Device ID Register—Offset 0x02 ....... 17-45
17.3.8.1.3 PCI Express Command Register—Offset 0x04 ....... 17-46
17.3.8.1.4 PCI Express Status Register—Offset 0x06 ....... 17-47
17.3.8.1.5 PCI Express Revision ID Register—Offset 0x08 ....... 17-48
17.3.8.1.6 PCI Express Class Code Register—Offset 0x09 ....... 17-49
17.3.8.1.7 PCI Express Cache Line Size Register—Offset 0x0C ....... 17-49
17.3.8.1.8 PCI Express Latency Timer Register—0x0D ....... 17-50
17.3.8.1.9 PCI Express Header Type Register—0x0E ....... 17-50
17.3.8.1.10 PCI Express BIST Register—0x0F ....... 17-51
17.3.8.2 Type 0 Configuration Header ....... 17-51
17.3.8.2.1 PCI Express Base Address Registers—0x10–0x27 ....... 17-51
17.3.8.2.2 PCI Express Subsystem Vendor ID Register (EP-Mode Only)—0x2C ....... 17-54

# Contents

Paragraph Number | Title | Page Number

17.3.8.2.3 PCI Express Subsystem ID Register (EP-Mode Only)—0x2E ....... 17-54
17.3.8.2.4 Capabilities Pointer Register—0x34 ....... 17-55
17.3.8.2.5 PCI Express Interrupt Line Register (EP-Mode Only)—0x3C ....... 17-55
17.3.8.2.6 PCI Express Interrupt Pin Register—0x3D ....... 17-56
17.3.8.2.7 PCI Express Minimum Grant Register (EP-Mode Only)—0x3E ....... 17-56
17.3.8.2.8 PCI Express Maximum Latency Register (EP-Mode Only)—0x3F ....... 17-57
17.3.8.3 Type 1 Configuration Header ....... 17-57
17.3.8.3.1 PCI Express Base Address Register 0—0x10 ....... 17-58
17.3.8.3.2 PCI Express Primary Bus Number Register—Offset 0x18 ....... 17-58
17.3.8.3.3 PCI Express Secondary Bus Number Register—Offset 0x19 ....... 17-59
17.3.8.3.4 PCI Express Subordinate Bus Number Register—Offset 0x1A ....... 17-59
17.3.8.3.5 PCI Express Secondary Latency Timer Register—0x1B ....... 17-60
17.3.8.3.6 PCI Express I/O Base Register—0x1C ....... 17-60
17.3.8.3.7 PCI Express I/O Limit Register—0x1D ....... 17-60
17.3.8.3.8 PCI Express Secondary Status Register—0x1E ....... 17-61
17.3.8.3.9 PCI Express Memory Base Register—0x20 ....... 17-62
17.3.8.3.10 PCI Express Memory Limit Register—0x22 ....... 17-62
17.3.8.3.11 PCI Express Prefetchable Memory Base Register—0x24 ....... 17-63
17.3.8.3.12 PCI Express Prefetchable Memory Limit Register—0x26 ....... 17-63
17.3.8.3.13 PCI Express Prefetchable Base Upper 32 Bits Register—0x28 ....... 17-64
17.3.8.3.14 PCI Express Prefetchable Limit Upper 32 Bits Register—0x2C ....... 17-64
17.3.8.3.15 PCI Express I/O Base Upper 16 Bits Register—0x30 ....... 17-64
17.3.8.3.16 PCI Express I/O Limit Upper 16 Bits Register—0x32 ....... 17-65
17.3.8.3.17 Capabilities Pointer Register—0x34 ....... 17-65
17.3.8.3.18 PCI Express Interrupt Line Register—0x3C ....... 17-66
17.3.8.3.19 PCI Express Interrupt Pin Register—0x3D ....... 17-66
17.3.8.3.20 PCI Express Bridge Control Register—0x3E ....... 17-67
17.3.9 PCI Compatible Device-Specific Configuration Space ....... 17-68
17.3.9.1 PCI Express Power Management Capability ID Register—0x44 ....... 17-69
17.3.9.2 PCI Express Power Management Capabilities Register—0x46 ....... 17-69
17.3.9.3 PCI Express Power Management Status and Control Register—0x48 ....... 17-70
17.3.9.4 PCI Express Power Management Data Register—0x4B ....... 17-70
17.3.9.5 PCI Express Capability ID Register—0x4C ....... 17-71
17.3.9.6 PCI Express Capabilities Register—0x4E ....... 17-71
17.3.9.7 PCI Express Device Capabilities Register—0x50 ....... 17-72
17.3.9.8 PCI Express Device Control Register—0x54 ....... 17-72
17.3.9.9 PCI Express Device Status Register—0x56 ....... 17-73
17.3.9.10 PCI Express Link Capabilities Register—0x58 ....... 17-74
17.3.9.11 PCI Express Link Control Register—0x5C ....... 17-74
17.3.9.12 PCI Express Link Status Register—0x5E ....... 17-75
17.3.9.13 PCI Express Slot Capabilities Register—0x60 ....... 17-76

# Contents

| Paragraph Number | Title | Page Number |
|---|---|---|
| 17.3.9.14 | PCI Express Slot Control Register—0x64 | 17-77 |
| 17.3.9.15 | PCI Express Slot Status Register—0x66 | 17-77 |
| 17.3.9.16 | PCI Express Root Control Register (RC Mode Only)—0x68 | 17-78 |
| 17.3.9.17 | PCI Express Root Status Register (RC Mode Only)—0x6C | 17-78 |
| 17.3.9.18 | PCI Express MSI Message Capability ID Register (EP Mode Only)—0x70 | 17-79 |
| 17.3.9.19 | PCI Express MSI Message Control Register (EP Mode Only)—0x72 | 17-80 |
| 17.3.9.20 | PCI Express MSI Message Address Register (EP Mode Only)—0x74 | 17-80 |
| 17.3.9.21 | PCI Express MSI Message Upper Address Register (EP Mode Only)—0x78 | 17-81 |
| 17.3.9.22 | PCI Express MSI Message Data Register (EP Mode Only)—0x7C | 17-81 |
| 17.3.10 | PCI Express Extended Configuration Space | 17-82 |
| 17.3.10.1 | PCI Express Advanced Error Reporting Capability ID Register—0x100 | 17-83 |
| 17.3.10.2 | PCI Express Uncorrectable Error Status Register—0x104 | 17-83 |
| 17.3.10.3 | PCI Express Uncorrectable Error Mask Register—0x108 | 17-84 |
| 17.3.10.4 | PCI Express Uncorrectable Error Severity Register—0x10C | 17-85 |
| 17.3.10.5 | PCI Express Correctable Error Status Register—0x110 | 17-86 |
| 17.3.10.6 | PCI Express Correctable Error Mask Register—0x114 | 17-86 |
| 17.3.10.7 | PCI Express Advanced Error Capabilities and Control Register—0x118 | 17-87 |
| 17.3.10.8 | PCI Express Header Log Register—0x11C–0x12B | 17-88 |
| 17.3.10.9 | PCI Express Root Error Command Register—0x12C | 17-89 |
| 17.3.10.10 | PCI Express Root Error Status Register—0x130 | 17-89 |
| 17.3.10.11 | PCI Express Correctable Error Source ID Register—0x134 | 17-90 |
| 17.3.10.12 | PCI Express Error Source ID Register—0x136 | 17-90 |
| 17.3.10.13 | LTSSM State Status Register—0x404 | 17-91 |
| 17.3.10.14 | PCI Express Controller Core Clock Ratio Register—0x440 | 17-92 |
| 17.3.10.15 | PCI Express Power Management Timer Register—0x450 | 17-93 |
| 17.3.10.16 | PCI Express PME Time-Out Register (EP-Mode Only)—0x454 | 17-94 |
| 17.3.10.17 | PCI Express Subsystem Vendor ID Update Register (EP Mode Only)—0x478 | 17-95 |
| 17.3.10.18 | Configuration Ready Register—0x4B0 | 17-95 |
| 17.3.10.19 | PME_To_Ack Timeout Register (RC-Mode Only)—0x590 | 17-96 |
| 17.3.10.20 | Secondary Status Interrupt Mask Register (RC-Mode Only)—0x5A0 | 17-96 |
| 17.4 | Functional Description | 17-97 |
| 17.4.1 | Architecture | 17-98 |
| 17.4.1.1 | PCI Express Transactions | 17-99 |
| 17.4.1.2 | Byte Ordering | 17-99 |
| 17.4.1.2.1 | Byte Order for Configuration Transactions | 17-101 |
| 17.4.1.3 | Lane Reversal | 17-101 |
| 17.4.1.4 | Transaction Ordering Rules | 17-102 |
| 17.4.1.5 | Memory Space Addressing | 17-102 |
| 17.4.1.6 | I/O Space Addressing | 17-102 |

# Contents

Paragraph Number | Title | Page Number

17.4.1.7 Configuration Space Addressing ............ 17-103
17.4.1.8 Serialization of Configuration and I/O Writes ............ 17-103
17.4.1.9 Messages ............ 17-103
17.4.1.9.1 Outbound ATMU Message Generation ............ 17-103
17.4.1.9.2 Inbound Messages ............ 17-105
17.4.1.10 Error Handling ............ 17-107
17.4.1.10.1 PCI Express Error Logging and Signaling ............ 17-107
17.4.1.10.2 PCI Express Controller Internal Interrupt Sources ............ 17-109
17.4.1.10.3 Error Conditions ............ 17-110
17.4.2 Interrupts ............ 17-113
17.4.2.1 EP Interrupt Generation ............ 17-113
17.4.2.1.1 Hardware INTx Message Generation ............ 17-113
17.4.2.1.2 Hardware MSI Generation ............ 17-113
17.4.2.1.3 Software INTx Message Generation ............ 17-113
17.4.2.1.4 Software MSI Generation ............ 17-113
17.4.2.2 RC Handling of INTx Message and MSI Interrupts ............ 17-114
17.4.2.2.1 INTx Message Handling ............ 17-114
17.4.2.2.2 MSI Handling ............ 17-114
17.4.3 Initial Credit Advertisement ............ 17-114
17.4.4 Power Management ............ 17-115
17.4.4.1 L2/L3 Ready Link State ............ 17-115
17.4.5 Hot Reset ............ 17-116
17.4.6 Link Down ............ 17-116
17.5 Initialization/Application Information ............ 17-116
17.5.1 Boot Mode and Inbound Configuration Transactions ............ 17-116

## Chapter 18
## Enhanced Serial Peripheral Interface

18.1 Introduction ............ 18-1
18.1.1 Features ............ 18-2
18.1.2 eSPI Transmission and Reception process ............ 18-2
18.1.3 Modes of Operation ............ 18-3
18.2 External Signal Descriptions ............ 18-3
18.2.1 Overview ............ 18-4
18.2.2 Detailed Signal Descriptions ............ 18-4
18.3 Memory Map/Register Definition ............ 18-5
18.3.1 Register Descriptions ............ 18-6
18.3.1.1 eSPI Mode Register (SPMODE) ............ 18-6
18.3.1.2 eSPI Event Register (SPIE) ............ 18-6
18.3.1.3 eSPI Mask Register (SPIM) ............ 18-7

# Contents

Paragraph Number | Title | Page Number

18.3.1.4 eSPI Command Register (SPCOM) .......... 18-9
18.3.1.5 eSPI Transmit FIFO Access Register (SPITF) .......... 18-10
18.3.1.6 eSPI Receive FIFO Access Register (SPIRF) .......... 18-11
18.3.1.7 eSPI CS*n* Mode Registers (SPMODE0–4) .......... 18-12
18.3.1.8 CI and CP Values for Various eSPI Devices .......... 18-14
18.4 eSPI Programming Examples .......... 18-14
18.4.1 24-bit Address Example .......... 18-14
18.4.2 16-bit Address Example .......... 18-15

## Chapter 19
## SATA Controller

19.1 Overview .......... 19-1
19.2 Command Operation .......... 19-2
19.2.1 Command Issue .......... 19-2
19.2.2 Command Service .......... 19-3
19.2.3 Command Completion Interrupt Timing .......... 19-3
19.2.4 DMA Context (Read Data) .......... 19-3
19.2.5 DMA Context (Write Data) .......... 19-3
19.2.6 DMAT Primitive Processing .......... 19-3
19.3 Command Layer Overview .......... 19-3
19.3.1 SATA Memory Map/Register Definition .......... 19-4
19.3.2 Command Registers .......... 19-5
19.3.2.1 Command Queue Register (CQR) .......... 19-5
19.3.2.2 Command Active Register (CAR) .......... 19-6
19.3.2.3 Command Completed Register (CCR) .......... 19-6
19.3.2.4 Command Error Register (CER) .......... 19-7
19.3.2.5 Device Error Register (DER) .......... 19-8
19.3.2.6 Command Header Base Address Register (CHBA) .......... 19-8
19.3.2.7 Host Status Register (HStatus) .......... 19-9
19.3.2.7.1 Error Processing .......... 19-11
19.3.2.8 Host Control Register (HControl) .......... 19-12
19.3.2.8.1 Bringing the SATA Controller Online/Offline .......... 19-13
19.3.2.9 Port Number Queue Register (CQPMP) .......... 19-13
19.3.2.10 Signature Register (SIG) .......... 19-14
19.3.2.11 Interrupt Coalescing Control Register (ICC) .......... 19-14
19.3.3 SATA Superset Registers .......... 19-15
19.3.3.1 SATA Interface Status Register (SStatus) .......... 19-15
19.3.3.2 SATA Interface Error Register (SError) .......... 19-16
19.3.3.3 SATA Interface Control Register (SControl) .......... 19-18
19.3.3.4 SATA Interface Notification Register (SNotification) .......... 19-19

# Contents

Paragraph Number | Title | Page Number

19.3.4 Control Status Registers .......... 19-20
19.3.4.1 Transport Layer Configuration Register (TransCfg) .......... 19-20
19.3.4.2 Transport Layer Status Register (TransStatus) .......... 19-21
19.3.4.3 Link Layer Configuration Register (LinkCfg) .......... 19-21
19.3.4.4 Link Layer Configuration Register1 (LinkCfg1) .......... 19-22
19.3.4.5 Link Layer Configuration Register2 (LinkCfg2) .......... 19-23
19.3.4.6 Link Layer Status Register (LinkStatus) .......... 19-23
19.3.4.7 Link Layer Status Register1 (LinkStatus1) .......... 19-24
19.3.4.8 PHY Control Configuration Register1 (PhyCtrlCfg1) .......... 19-26
19.3.4.9 Link Layer Command Status Register (CommandStatus) .......... 19-27
19.3.5 System Control Registers .......... 19-28
19.3.5.1 System Priority Register (SYSPR) .......... 19-28
19.3.6 Command Header .......... 19-29
19.3.7 Command Descriptor .......... 19-31
19.3.7.1 Command FIS Non-Queued Commands (CFIS) .......... 19-31
19.3.7.2 Command FIS First Party DMA Commands NCQ .......... 19-32
19.3.7.3 Status FIS (SFIS) .......... 19-32
19.3.7.4 ATAPI Command (ACMD) .......... 19-32
19.3.7.5 Physical Region Descriptor Table (PRDT) .......... 19-33
19.3.8 Vendor-Specific BIST Operation .......... 19-33
19.4 Transport Layer Architectural Overview .......... 19-35
19.5 Link Layer Overview .......... 19-36
19.5.1 Link Layer functionality .......... 19-36
19.5.1.1 Link Idle State Machine .......... 19-37
19.5.1.2 Transmit State Machine .......... 19-37
19.5.1.3 Receive State Machine .......... 19-38
19.5.1.4 Power Mode Change State Machine .......... 19-38
19.5.1.5 Frame Content Scrambler and Descrambler .......... 19-39
19.5.1.6 CRC Generator and Checker .......... 19-39
19.5.1.7 8B/10B Encode and Decode .......... 19-39
19.5.1.8 CONT Primitive Processing .......... 19-40
19.5.1.9 ALIGN Insertion .......... 19-40
19.5.1.10 Debug Functionality .......... 19-40
19.5.1.11 BIST Support .......... 19-41
19.6 PHY Control Layer Overview .......... 19-41
19.7 Initialization/Application Information .......... 19-41
19.7.1 SATA Controller Initialization Steps .......... 19-41

## Chapter 20
## Enhanced Secure Digital Host Controller

# Contents

Paragraph Number | Title | Page Number

20.1 Overview ..................... 20-1
20.2 Features ..................... 20-2
20.2.1 Data Transfer Modes ..................... 20-3
20.3 External Signal Description ..................... 20-3
20.4 Memory Map/Register Definition ..................... 20-5
20.4.1 DMA System Address Register (DSADDR) ..................... 20-6
20.4.2 Block Attributes Register (BLKATTR) ..................... 20-6
20.4.3 Command Argument Register (CMDARG) ..................... 20-7
20.4.4 Transfer Type Register (XFERTYP) ..................... 20-8
20.4.5 Command Response 0–3 (CMDRSP0–3) ..................... 20-11
20.4.6 Buffer Data Port Register (DATPORT) ..................... 20-12
20.4.7 Present State Register (PRSSTAT) ..................... 20-13
20.4.8 Protocol Control Register (PROCTL) ..................... 20-17
20.4.9 System Control Register (SYSCTL) ..................... 20-19
20.4.10 Interrupt Status Register (IRQSTAT) ..................... 20-22
20.4.11 Interrupt Status Enable Register (IRQSTATEN) ..................... 20-26
20.4.12 Interrupt Signal Enable Register (IRQSIGEN) ..................... 20-29
20.4.13 Auto CMD12 Error Status Register (AUTOC12ERR) ..................... 20-31
20.4.14 Host Controller Capabilities (HOSTCAPBLT) ..................... 20-33
20.4.15 Watermark Level Register (WML) ..................... 20-34
20.4.16 Force Event Register (FEVT) ..................... 20-34
20.4.17 Host Controller Version Register (HOSTVER) ..................... 20-36
20.4.18 DMA Control Register (DCR) ..................... 20-37
20.5 Functional Description ..................... 20-37
20.5.1 Data Buffer ..................... 20-38
20.5.1.1 Write Operation Sequence ..................... 20-38
20.5.1.2 Read Operation Sequence ..................... 20-39
20.5.1.3 Data Buffer Size ..................... 20-39
20.5.2 DMA CCB Interface ..................... 20-40
20.5.2.1 Internal DMA Request ..................... 20-40
20.5.2.2 DMA Burst Length ..................... 20-40
20.5.2.3 CCB Master Interface ..................... 20-41
20.5.3 SD Protocol Unit ..................... 20-41
20.5.3.1 SD Transceiver ..................... 20-41
20.5.3.2 SD Clock and Monitor ..................... 20-41
20.5.3.3 Command Agent ..................... 20-42
20.5.3.4 Data Agent ..................... 20-42
20.5.4 Clock & Reset Manager ..................... 20-42
20.5.5 Clock Generator ..................... 20-43
20.5.6 Card Insertion and Removal Detection ..................... 20-43
20.5.7 Power Management and Wake-Up Events ..................... 20-43

# Contents

Paragraph Number | Title | Page Number

20.5.7.1 Setting Wake Up Events ........ 20-44
20.6 Initialization/Application Information ........ 20-44
20.6.1 Command Send and Response Receive Basic Operation ........ 20-44
20.6.2 Card Identification Mode ........ 20-45
20.6.2.1 Card Detect ........ 20-45
20.6.2.2 Reset ........ 20-46
20.6.2.3 Voltage Validation ........ 20-47
20.6.2.4 Card Registry ........ 20-47
20.6.3 Card Access ........ 20-48
20.6.3.1 Block Write ........ 20-48
20.6.3.1.1 Normal Write ........ 20-49
20.6.3.1.2 Write with Pause ........ 20-50
20.6.3.2 Block Read ........ 20-51
20.6.3.2.1 Normal Read ........ 20-51
20.6.3.2.2 Read with Pause ........ 20-51
20.6.3.3 Transfer Error ........ 20-52
20.6.3.3.1 CRC Error ........ 20-52
20.6.3.3.2 Internal DMA Error ........ 20-52
20.6.3.3.3 Auto CMD12 Error ........ 20-53
20.6.3.4 Card Interrupt ........ 20-53
20.6.4 Switch Function ........ 20-53
20.6.4.1 Query, Enable and Disable SD High Speed Mode ........ 20-53
20.6.4.2 Query, Enable and Disable MMC High Speed Mode ........ 20-54
20.6.4.3 Set MMC Bus Width ........ 20-55
20.6.5 Commands for MMC/SD ........ 20-55
20.6.6 Software Restrictions ........ 20-60

## Chapter 21
## Universal Serial Bus Interfaces

21.1 Introduction ........ 21-1
21.1.1 Overview ........ 21-2
21.1.2 Features ........ 21-2
21.1.3 Modes of Operation ........ 21-2
21.2 External Signals ........ 21-2
21.2.1 ULPI Interface ........ 21-2
21.2.2 PHY Clocks ........ 21-4
21.3 Memory Map/Register Definitions ........ 21-4
21.3.1 Capability Registers ........ 21-6
21.3.1.1 Capability Registers Length (CAPLENGTH) ........ 21-7
21.3.1.2 Host Controller Interface Version (HCIVERSION) ........ 21-7

# Contents

Paragraph Number | Title | Page Number

21.3.1.3 Host Controller Structural Parameters (HCSPARAMS) ...................................... 21-7
21.3.1.4 Host Controller Capability Parameters (HCCPARAMS) ..................................... 21-8
21.3.1.5 Device Controller Interface Version (DCIVERSION)—Non-EHCI ..................... 21-9
21.3.1.6 Device Controller Capability Parameters (DCCPARAMS)—Non-EHCI ........... 21-10
21.3.2 Operational Registers ........................................................................................ 21-10
21.3.2.1 USB Command Register (USBCMD) ............................................................... 21-11
21.3.2.2 USB Status Register (USBSTS) ......................................................................... 21-13
21.3.2.3 USB Interrupt Enable Register (USBINTR) ...................................................... 21-15
21.3.2.4 Frame Index Register (FRINDEX) ...................................................................... 21-17
21.3.2.5 Control Data Structure Segment Register (CTRLDSSEGMENT) ..................... 21-18
21.3.2.6 Periodic Frame List Base Address Register (PERIODICLISTBASE) ................ 21-18
21.3.2.7 Device Address Register (DEVICEADDR)—Non-EHCI .................................. 21-19
21.3.2.8 Current Asynchronous List Address Register (ASYNCLISTADDR) ................ 21-19
21.3.2.9 Endpoint List Address Register (ENDPOINTLISTADDR)—Non-EHCI .......... 21-20
21.3.2.10 Master Interface Data Burst Size Register (BURSTSIZE)—Non-EHCI ............ 21-21
21.3.2.11 Transmit FIFO Tuning Controls Register (TXFILLTUNING)—Non-EHCI ...... 21-21
21.3.2.12 ULPI Register Access (ULPI VIEWPORT) ........................................................ 21-23
21.3.2.13 Configure Flag Register (CONFIGFLAG) .......................................................... 21-24
21.3.2.14 Port Status and Control Register (PORTSC) ...................................................... 21-25
21.3.2.15 USB Mode Register (USBMODE)—Non-EHCI ................................................ 21-29
21.3.2.16 Endpoint Setup Status Register (ENDPTSETUPSTAT)—Non-EHCI ................ 21-30
21.3.2.17 Endpoint Initialization Register (ENDPTPRIME)—Non-EHCI ......................... 21-31
21.3.2.18 Endpoint Flush Register (ENDPTFLUSH)—Non-EHCI ................................... 21-32
21.3.2.19 Endpoint Status Register (ENDPTSTATUS)—Non-EHCI ................................ 21-32
21.3.2.20 Endpoint Complete Register (ENDPTCOMPLETE)—Non-EHCI ..................... 21-33
21.3.2.21 Endpoint Control Register 0 (ENDPTCTRL0)—Non-EHCI ............................. 21-33
21.3.2.22 Endpoint Control Register *n* (ENDPTCTRL*n*)—Non-EHCI ............................. 21-35
21.3.2.23 SNOOP1 and SNOOP2—Non-EHCI .................................................................. 21-36
21.3.2.24 Age Count Threshold Register (AGE_CNT_THRESH)—Non-EHCI ............... 21-37
21.3.2.25 Priority Control Register (PRI_CTRL)—Non-EHCI .......................................... 21-38
21.3.2.26 System Interface Control Register (SI_CTRL)—Non-EHCI ............................. 21-39
21.3.2.27 USB General Purpose Register (CONTROL)—Non-EHCI ............................... 21-39
21.4 Functional Description ........................................................................................... 21-40
21.4.1 System Interface ................................................................................................ 21-40
21.4.2 DMA Engine ...................................................................................................... 21-40
21.4.3 FIFO RAM Controller ....................................................................................... 21-41
21.4.4 PHY Interface .................................................................................................... 21-41
21.5 Host Data Structures .............................................................................................. 21-41
21.5.1 Periodic Frame List ........................................................................................... 21-42
21.5.2 Asynchronous List Queue Head Pointer ............................................................ 21-43
21.5.3 Isochronous (High-Speed) Transfer Descriptor (iTD) ........................................ 21-44

# Contents

| Paragraph Number | Title | Page Number |
|---|---|---|
| 21.5.3.1 | Next Link Pointer | 21-44 |
| 21.5.3.2 | iTD Transaction Status and Control List | 21-45 |
| 21.5.3.3 | iTD Buffer Page Pointer List (Plus) | 21-46 |
| 21.5.4 | Split Transaction Isochronous Transfer Descriptor (siTD) | 21-47 |
| 21.5.4.1 | Next Link Pointer | 21-48 |
| 21.5.4.2 | siTD Endpoint Capabilities/Characteristics | 21-48 |
| 21.5.4.3 | siTD Transfer State | 21-49 |
| 21.5.4.4 | siTD Buffer Pointer List (Plus) | 21-50 |
| 21.5.4.5 | siTD Back Link Pointer | 21-51 |
| 21.5.5 | Queue Element Transfer Descriptor (qTD) | 21-51 |
| 21.5.5.1 | Next qTD Pointer | 21-52 |
| 21.5.5.2 | Alternate Next qTD Pointer | 21-52 |
| 21.5.5.3 | qTD Token | 21-53 |
| 21.5.5.4 | qTD Buffer Page Pointer List | 21-56 |
| 21.5.6 | Queue Head | 21-56 |
| 21.5.6.1 | Queue Head Horizontal Link Pointer | 21-57 |
| 21.5.6.2 | Endpoint Capabilities/Characteristics | 21-57 |
| 21.5.6.3 | Transfer Overlay | 21-59 |
| 21.5.7 | Periodic Frame Span Traversal Node (FSTN) | 21-60 |
| 21.5.7.1 | FTSN Normal Path Pointer | 21-61 |
| 21.5.7.2 | FSTN Back Path Link Pointer | 21-61 |
| 21.6 | Host Operations | 21-61 |
| 21.6.1 | Host Controller Initialization | 21-62 |
| 21.6.2 | Power Port | 21-63 |
| 21.6.3 | Reporting Over-Current | 21-63 |
| 21.6.4 | Suspend/Resume | 21-63 |
| 21.6.4.1 | Port Suspend/Resume | 21-64 |
| 21.6.5 | Schedule Traversal Rules | 21-66 |
| 21.6.6 | Periodic Schedule Frame Boundaries vs. Bus Frame Boundaries | 21-67 |
| 21.6.7 | Periodic Schedule | 21-69 |
| 21.6.8 | Managing Isochronous Transfers Using iTDs | 21-70 |
| 21.6.8.1 | Host Controller Operational Model for iTDs | 21-70 |
| 21.6.8.2 | Software Operational Model for iTDs | 21-72 |
| 21.6.8.2.1 | Periodic Scheduling Threshold | 21-73 |
| 21.6.9 | Asynchronous Schedule | 21-74 |
| 21.6.9.1 | Adding Queue Heads to Asynchronous Schedule | 21-75 |
| 21.6.9.2 | Removing Queue Heads from Asynchronous Schedule | 21-76 |
| 21.6.9.3 | Empty Asynchronous Schedule Detection | 21-78 |
| 21.6.9.4 | Asynchronous Schedule Traversal: Start Event | 21-79 |
| 21.6.9.5 | Reclamation Status Bit (USBSTS Register) | 21-79 |
| 21.6.10 | Managing Control/Bulk/Interrupt Transfers via Queue Heads | 21-79 |

# Contents

Paragraph Number | Title | Page Number

21.6.10.1 Buffer Pointer List Use for Data Streaming with qTDs ..... 21-80
21.6.10.2 Adding Interrupt Queue Heads to the Periodic Schedule ..... 21-82
21.6.10.3 Managing Transfer Complete Interrupts from Queue Heads ..... 21-82
21.6.11 Ping Control ..... 21-83
21.6.12 Split Transactions ..... 21-84
21.6.12.1 Split Transactions for Asynchronous Transfers ..... 21-84
21.6.12.1.1 Asynchronous—Do-Start-Split ..... 21-85
21.6.12.1.2 Asynchronous—Do-Complete-Split ..... 21-85
21.6.12.2 Split Transaction Interrupt ..... 21-86
21.6.12.2.1 Split Transaction Scheduling Mechanisms for Interrupt ..... 21-86
21.6.12.2.2 Host Controller Operational Model for FSTNs ..... 21-89
21.6.12.2.3 Software Operational Model for FSTNs ..... 21-91
21.6.12.2.4 Tracking Split Transaction Progress for Interrupt Transfers ..... 21-92
21.6.12.2.5 Split Transaction Execution State Machine for Interrupt ..... 21-92
21.6.12.2.6 Periodic Interrupt—Do-Start-Split ..... 21-93
21.6.12.2.7 Periodic Interrupt—Do-Complete-Split ..... 21-94
21.6.12.2.8 Managing the QH[FrameTag] Field ..... 21-97
21.6.12.2.9 Rebalancing the Periodic Schedule ..... 21-98
21.6.12.3 Split Transaction Isochronous ..... 21-98
21.6.12.3.1 Split Transaction Scheduling Mechanisms for Isochronous ..... 21-99
21.6.12.3.2 Tracking Split Transaction Progress for Isochronous Transfers ..... 21-102
21.6.12.3.3 Split Transaction Execution State Machine for Isochronous ..... 21-104
21.6.12.3.4 Periodic Isochronous—Do-Start-Split ..... 21-104
21.6.12.3.5 Periodic Isochronous—Do Complete Split ..... 21-106
21.6.12.3.6 Complete-Split for Scheduling Boundary Cases 2a, 2b ..... 21-109
21.6.12.3.7 Split Transaction for Isochronous—Processing Example ..... 21-110
21.6.13 Port Test Modes ..... 21-111
21.6.14 Interrupts ..... 21-112
21.6.14.1 Transfer/Transaction Based Interrupts ..... 21-113
21.6.14.1.1 Transaction Error ..... 21-113
21.6.14.1.2 Serial Bus Babble ..... 21-113
21.6.14.1.3 Data Buffer Error ..... 21-114
21.6.14.1.4 USB Interrupt (Interrupt on Completion (IOC)) ..... 21-115
21.6.14.1.5 Short Packet ..... 21-115
21.6.14.2 Host Controller Event Interrupts ..... 21-115
21.6.14.2.1 Port Change Events ..... 21-115
21.6.14.2.2 Frame List Rollover ..... 21-115
21.6.14.2.3 Interrupt on Async Advance ..... 21-115
21.6.14.2.4 Host System Error ..... 21-116
21.7 Device Data Structures ..... 21-116
21.7.1 Endpoint Queue Head ..... 21-117

# Contents

| Paragraph Number | Title | Page Number |
|---|---|---|
| 21.7.1.1 | Endpoint Capabilities/Characteristics | 21-118 |
| 21.7.1.2 | Transfer Overlay | 21-119 |
| 21.7.1.3 | Current dTD Pointer | 21-119 |
| 21.7.1.4 | Set-up Buffer | 21-119 |
| 21.7.2 | Endpoint Transfer Descriptor (dTD) | 21-120 |
| 21.8 | Device Operational Model | 21-122 |
| 21.8.1 | Device Controller Initialization | 21-122 |
| 21.8.2 | Port State and Control | 21-123 |
| 21.8.2.1 | Bus Reset | 21-125 |
| 21.8.2.2 | Suspend/Resume | 21-126 |
| 21.8.2.2.1 | Suspend Description | 21-126 |
| 21.8.2.2.2 | Suspend Operational Model | 21-126 |
| 21.8.2.2.3 | Resume | 21-126 |
| 21.8.3 | Managing Endpoints | 21-126 |
| 21.8.3.1 | Endpoint Initialization | 21-127 |
| 21.8.3.1.1 | Stalling | 21-127 |
| 21.8.3.2 | Data Toggle | 21-128 |
| 21.8.3.2.1 | Data Toggle Reset | 21-128 |
| 21.8.3.2.2 | Data Toggle Inhibit | 21-128 |
| 21.8.3.3 | Device Operational Model For Packet Transfers | 21-129 |
| 21.8.3.3.1 | Priming Transmit Endpoints | 21-129 |
| 21.8.3.3.2 | Priming Receive Endpoints | 21-129 |
| 21.8.3.4 | Interrupt/Bulk Endpoint Operational Model | 21-130 |
| 21.8.3.4.1 | Interrupt/Bulk Endpoint Bus Response Matrix | 21-131 |
| 21.8.3.5 | Control Endpoint Operation Model | 21-132 |
| 21.8.3.5.1 | Setup Phase | 21-132 |
| 21.8.3.5.2 | Data Phase | 21-132 |
| 21.8.3.5.3 | Status Phase | 21-133 |
| 21.8.3.5.4 | Control Endpoint Bus Response Matrix | 21-133 |
| 21.8.3.6 | Isochronous Endpoint Operational Model | 21-134 |
| 21.8.3.6.1 | Isochronous Pipe Synchronization | 21-135 |
| 21.8.3.6.2 | Isochronous Endpoint Bus Response Matrix | 21-136 |
| 21.8.4 | Managing Queue Heads | 21-136 |
| 21.8.4.1 | Queue Head Initialization | 21-137 |
| 21.8.4.2 | Operational Model for Setup Transfers | 21-137 |
| 21.8.5 | Managing Transfers with Transfer Descriptors | 21-138 |
| 21.8.5.1 | Software Link Pointers | 21-138 |
| 21.8.5.2 | Building a Transfer Descriptor | 21-138 |
| 21.8.5.3 | Executing a Transfer Descriptor | 21-139 |
| 21.8.5.4 | Transfer Completion | 21-139 |
| 21.8.5.5 | Flushing/De-Priming an Endpoint | 21-140 |

# Contents

Paragraph Number | Title | Page Number

21.8.5.6 Device Error Matrix .......... 21-140
21.8.6 Servicing Interrupts .......... 21-141
21.8.6.1 High-Frequency Interrupts .......... 21-141
21.8.6.2 Low-Frequency Interrupts .......... 21-141
21.8.6.3 Error Interrupts .......... 21-142
21.9 Deviations from the EHCI Specifications .......... 21-142
21.9.1 Embedded Transaction Translator Function .......... 21-142
21.9.1.1 Capability Registers .......... 21-143
21.9.1.2 Operational Registers .......... 21-143
21.9.1.3 Discovery .......... 21-143
21.9.1.4 Data Structures .......... 21-143
21.9.1.5 Operational Model .......... 21-144
21.9.1.5.1 Microframe Pipeline .......... 21-144
21.9.1.5.2 Split State Machines .......... 21-145
21.9.1.5.3 Asynchronous Transaction Scheduling and Buffer Management .......... 21-145
21.9.1.5.4 Periodic Transaction Scheduling and Buffer Management .......... 21-145
21.9.1.5.5 Multiple Transaction Translators .......... 21-146
21.9.2 Device Operation .......... 21-146
21.9.3 Non-Zero Fields the Register File .......... 21-146
21.9.4 SOF Interrupt .......... 21-146
21.9.5 Embedded Design .......... 21-146
21.9.5.1 Frame Adjust Register .......... 21-147
21.9.6 Miscellaneous Variations from EHCI .......... 21-147
21.9.6.1 Programmable Physical Interface Behavior .......... 21-147
21.9.6.2 Discovery .......... 21-147
21.9.6.2.1 Port Reset .......... 21-147
21.9.6.2.2 Port Speed Detection .......... 21-147
21.10 Timing Diagrams .......... 21-148

## Chapter 22
## General Purpose I/O (GPIO)

22.1 Introduction .......... 22-1
22.1.1 Overview .......... 22-1
22.1.2 Features .......... 22-1
22.2 External Signal Description .......... 22-2
22.2.1 Signals Overview .......... 22-2
22.3 Memory Map/Register Definition .......... 22-2
22.3.1 GPIO Direction Register (GPDIR) .......... 22-2
22.3.2 GPIO Open Drain Register (GPODR) .......... 22-3
22.3.3 GPIO Data Register (GPDAT) .......... 22-3

# Contents

Paragraph Number | Title | Page Number

22.3.4 GPIO Interrupt Event Register (GPIER) .......... 22-4
22.3.5 GPIO Interrupt Mask Register (GPIMR) .......... 22-4
22.3.6 GPIO Interrupt Control Register (GPICR) .......... 22-5

## Part IV
## Global Functions and Debug

## Chapter 23
## Global Utilities

23.1 Overview .......... 23-1
23.2 Global Utilities Features .......... 23-1
23.2.1 Power Management and Block Disables .......... 23-1
23.2.2 Accessing Current POR Configuration Settings .......... 23-1
23.2.3 Signal Multiplexing .......... 23-1
23.2.4 Clock Control .......... 23-1
23.3 External Signal Description .......... 23-2
23.3.1 Signals Overview .......... 23-2
23.3.2 Detailed Signal Descriptions .......... 23-2
23.4 Memory Map/Register Definition .......... 23-3
23.4.1 Register Descriptions .......... 23-5
23.4.1.1 POR PLL Status Register (PORPLLSR) .......... 23-5
23.4.1.2 POR Boot Mode Status Register (PORBMSR) .......... 23-6
23.4.1.3 POR I/O Impedance Status and Control Register (PORIMPSCR) .......... 23-8
23.4.1.4 POR Device Status Register (PORDEVSR) .......... 23-9
23.4.1.5 POR Debug Mode Status Register (PORDBGMSR) .......... 23-12
23.4.1.6 POR Device Status Register 2 (PORDEVSR2) .......... 23-13
23.4.1.7 General-Purpose POR Configuration Register (GPPORCR) .......... 23-13
23.4.1.8 General Configuration Register (GENCFGR) .......... 23-14
23.4.1.9 Alternate Function Signal Multiplex Control Register (PMUXCR) .......... 23-14
23.4.1.10 Device Disable Register (DEVDISR) .......... 23-16
23.4.1.11 Power Management Jog Control Register (PMJCR) .......... 23-19
23.4.1.12 Power Management Control and Status Register (POWMGTCSR) .......... 23-21
23.4.1.13 Power Management Reset Counters Configuration Register (PMRCCR) .......... 23-22
23.4.1.14 Power Management Power Down Counters Configuration Register (PMPDCCR) .......... 23-24
23.4.1.15 Power Management Clock Disable Register (PMCDR) .......... 23-25
23.4.1.16 Machine Check Summary Register (MCPSUMR) .......... 23-26
23.4.1.17 Reset Request Status and Control Register (RSTRSCR) .......... 23-27
23.4.1.18 Exception Reset Control Register (ECTRSTCR) .......... 23-28
23.4.1.19 Automatic Reset Status Register (AUTORSTSR) .......... 23-28

# Contents

Paragraph Number | Title | Page Number

23.4.1.20 Processor Version Register (PVR) ........ 23-29
23.4.1.21 System Version Register (SVR) ........ 23-30
23.4.1.22 Reset Control Register (RSTCR) ........ 23-30
23.4.1.23 LBC Voltage Select Control Register (LBCVSELCR) ........ 23-31
23.4.1.24 DDR Clock Disable Register (DDRCLKDR) ........ 23-32
23.4.1.25 Clock Out Control Register (CLKOCR) ........ 23-33
23.4.1.26 ECM Control Register (ECMCR) ........ 23-33
23.4.1.27 General Control Register (GCR) ........ 23-34
23.4.1.28 SerDes1 Control Register 0 (SRDS1CR0) ........ 23-35
23.4.1.29 SerDes1 Control Register 2 (SRDS1CR2) ........ 23-37
23.4.1.30 SerDes2 Control Register 0 (SRDS2CR0) ........ 23-39
23.4.1.31 SerDes2 Control Register 1 (SRDS2CR1) ........ 23-41
23.4.1.32 SerDes2 Control Register 2 (SRDS2CR2) ........ 23-42
23.4.1.33 SerDes2 Control Register 3 (SRDS2CR3) ........ 23-44
23.5 Functional Description ........ 23-47
23.5.1 Power Management Controller (PMC) ........ 23-47
23.5.1.1 Overview ........ 23-47
23.5.1.2 Relationship Between Core and Device Power Management States ........ 23-48
23.5.1.3 CKSTP_IN0/1 is Not Power Management ........ 23-48
23.5.1.4 Dynamic Power Management ........ 23-49
23.5.1.5 Shutting Down Unused Blocks ........ 23-49
23.5.1.6 Software-Controlled Power-Down States ........ 23-49
23.5.1.6.1 Doze Mode ........ 23-49
23.5.1.6.2 Nap Mode ........ 23-50
23.5.1.6.3 Sleep Mode ........ 23-50
23.5.1.6.4 Deep Sleep Mode ........ 23-50
23.5.1.6.5 Jog Mode ........ 23-51
23.5.1.7 Power Management Control Fields (e500) ........ 23-52
23.5.1.8 Power-Down Sequence Coordination ........ 23-53
23.5.1.9 Interrupts and Power Management (e500) ........ 23-54
23.5.1.9.1 Interrupts and Power Management Controlled by MSR[WE] (e500) ........ 23-54
23.5.1.9.2 Interrupts and Power Management Controlled by POWMGTCSR (e500) ........ 23-55
23.5.1.10 Snooping in Power-Down Modes (e500) ........ 23-55
23.5.1.11 Software Considerations for Power Management (e500) ........ 23-55
23.5.1.12 Requirements for Reaching and Recovering from Sleep State ........ 23-56
23.5.1.13 Requirements for Reaching and Recovering from Deep Sleep State ........ 23-56
23.5.1.14 Requirements for Generating Wake-Up Events ........ 23-57
23.5.1.14.1 USB ........ 23-57
23.5.1.14.2 GPIO ........ 23-58
23.5.1.14.3 Timer ........ 23-58
23.5.1.14.4 eTSEC Wake-on LAN—Magic Packet ........ 23-58

# Contents

Paragraph Number | Title | Page Number

23.5.1.14.5 eTSEC Wake-on-LAN—ARP (User Defined) Packet .......... 23-59
23.5.1.15 External Power Supply Control .......... 23-59
23.5.1.16 Low Power Considerations .......... 23-60
23.5.1.16.1 POWER_OK Input Signal .......... 23-60
23.5.1.16.2 POWER_EN Output Signal .......... 23-61
23.5.1.16.3 DPSLP Register Bit .......... 23-61
23.5.1.16.4 RST_DPSLP Register Bit .......... 23-61

## Chapter 24
## Device Performance Monitor

24.1 Introduction .......... 24-1
24.1.1 Overview .......... 24-2
24.1.2 Features .......... 24-3
24.2 Signal Descriptions .......... 24-3
24.3 Memory Map and Register Definition .......... 24-3
24.3.1 Register Summary .......... 24-3
24.3.2 Control Registers .......... 24-5
24.3.2.1 Performance Monitor Global Control Register (PMGC0) .......... 24-5
24.3.2.2 Performance Monitor Local Control Registers (PMLCA*n*, PMLCB*n*) .......... 24-6
24.3.3 Counter Registers .......... 24-10
24.3.3.1 Performance Monitor Counters (PMC0–PMC9) .......... 24-10
24.4 Functional Description .......... 24-11
24.4.1 Performance Monitor Interrupt .......... 24-11
24.4.2 Event Counting .......... 24-12
24.4.3 Threshold Events .......... 24-12
24.4.4 Chaining .......... 24-13
24.4.5 Triggering .......... 24-13
24.4.6 Burstiness Counting .......... 24-14
24.4.7 Performance Monitor Events .......... 24-16
24.4.8 Performance Monitor Examples .......... 24-27

## Chapter 25
## Debug Features and Watchpoint Facility

25.1 Introduction .......... 25-1
25.1.1 Overview .......... 25-1
25.1.2 Features .......... 25-3
25.1.3 Modes of Operation .......... 25-3
25.1.3.1 Local Bus (LBC) Debug Mode .......... 25-4
25.1.3.2 DDR SDRAM Interface Debug Modes .......... 25-4

# Contents

Paragraph Number | Title | Page Number

25.1.3.3 Watchpoint Monitor Modes ........ 25-4
25.1.3.4 Trace Buffer Modes ........ 25-4
25.2 External Signal Description ........ 25-5
25.2.1 Overview ........ 25-5
25.2.2 Detailed Signal Descriptions ........ 25-6
25.2.2.1 Debug Signals—Details ........ 25-6
25.2.2.2 Watchpoint Monitor Trigger Signals—Details ........ 25-7
25.2.2.3 Test Signals—Details ........ 25-8
25.3 Memory Map/Register Definition ........ 25-9
25.3.1 Watchpoint Monitor Register Descriptions ........ 25-10
25.3.1.1 Watchpoint Monitor Control Registers 0–1 (WMCR0, WMCR1) ........ 25-10
25.3.1.2 Watchpoint Monitor Address Register (WMAR) ........ 25-12
25.3.1.3 Watchpoint Monitor Address Mask Register (WMAMR) ........ 25-13
25.3.1.4 Watchpoint Monitor Transaction Mask Register (WMTMR) ........ 25-13
25.3.1.5 Watchpoint Monitor Status Register (WMSR) ........ 25-15
25.3.2 Trace Buffer Register Descriptions ........ 25-15
25.3.2.1 Trace Buffer Control Registers (TBCR0, TBCR1) ........ 25-15
25.3.2.2 Trace Buffer Address Register (TBAR) ........ 25-18
25.3.2.3 Trace Buffer Address Mask Register (TBAMR) ........ 25-18
25.3.2.4 Trace Buffer Transaction Mask Register (TBTMR) ........ 25-19
25.3.2.5 Trace Buffer Status Register (TBSR) ........ 25-19
25.3.2.6 Trace Buffer Access Control Register (TBACR) ........ 25-20
25.3.2.7 Trace Buffer Access Data High Register (TBADHR) ........ 25-21
25.3.2.8 Trace Buffer Access Data Register (TBADR) ........ 25-21
25.3.3 Context ID Registers ........ 25-22
25.3.3.1 Programmed Context ID Register (PCIDR) ........ 25-22
25.3.3.2 Current Context ID Register (CCIDR) ........ 25-23
25.3.4 Trigger Out Function ........ 25-23
25.3.4.1 Trigger Out Source Register (TOSR) ........ 25-23
25.4 Functional Description ........ 25-24
25.4.1 Source and Target ID ........ 25-24
25.4.2 DDR SDRAM Interface Debug ........ 25-25
25.4.2.1 Debug Information on Debug Pins ........ 25-25
25.4.2.2 Debug Information on ECC Pins ........ 25-25
25.4.3 Local Bus Interface Debug ........ 25-26
25.4.4 Watchpoint Monitor ........ 25-26
25.4.4.1 Watchpoint Monitor Performance Monitor Events ........ 25-26
25.4.5 Trace Buffer ........ 25-27
25.4.5.1 Traced Data Formats (as a Function of TBCR1[IFSEL]) ........ 25-27
25.5 Initialization ........ 25-30

# Contents

Paragraph Number | Title | Page Number

## Appendix A
## Complete List of Configuration, Control, and Status Registers

A.1 General Utilities .......... A-1
A.1.1 Local Configuration Control .......... A-1
A.1.2 Local Access Windows .......... A-1
A.1.3 e500 Coherency Module (ECM) .......... A-2
A.1.4 DDR Memory Controller .......... A-3
A.1.5 I$^2$C Controllers .......... A-5
A.1.6 DUART .......... A-5
A.1.7 Enhanced Local Bus Controller .......... A-6
A.1.8 Enhanced Serial Peripheral Interface (eSPI) .......... A-8
A.1.9 PCI Controller .......... A-8
A.1.10 PCI Express Controllers .......... A-11
A.1.11 General-Purpose I/O (GPIO) .......... A-15
A.1.12 Serial ATA Controllers .......... A-16
A.1.13 L2 Cache/SRAM .......... A-17
A.1.14 DMA Controller .......... A-18
A.1.15 USB Registers .......... A-18
A.1.16 eTSEC Registers .......... A-20
A.1.17 eSDHC Registers .......... A-30
A.1.18 SEC Registers .......... A-31
A.2 Programmable Interrupt Controller (PIC) .......... A-37
A.2.1 PIC—Global Registers .......... A-37
A.2.2 PIC—Interrupt Source Registers .......... A-41
A.2.3 PIC—Processor (per-CPU) Registers .......... A-47
A.3 Device-Specific Utilities .......... A-48
A.3.1 Global Utilities .......... A-48
A.3.2 Device Performance Monitor .......... A-50
A.3.3 Watchpoint Monitor and Trace Buffer .......... A-51

## Appendix B
## Revision History

B.1 Changes From Revision 0 to Revision 1 .......... B-1

## Appendix C
## MPC8535E

C.1 Overview of Differences .......... C-1
C.2 Signal Differences .......... C-2

# Contents

Paragraph Number | Title | Page Number

C.3 Reset/Configuration Differences .......... C-3
C.3.1 Boot ROM Location .......... C-3
C.3.2 Host/Agent Configuration .......... C-4
C.3.3 I/O Port Selection .......... C-4
C.3.3.1 SerDes1 (PCI Express) I/O Port Selection .......... C-4
C.3.3.2 SerDes2 (SATA) I/O Port Selection .......... C-5
C.4 Differences in Peripheral Blocks .......... C-5
C.4.1 PCI Express Interfaces .......... C-5
C.4.2 USB Controllers .......... C-5
C.4.3 SATA Controllers .......... C-6
C.4.4 eTSEC Controllers .......... C-6

# Contents

| Paragraph<br>Number | Title | Page<br>Number |
|---|---|---|

# Figures

Figure Number | Title | Page Number

1-1 MPC8536E Block Diagram ..... 1-2
1-2 Multi-Function Printer Application ..... 1-11
1-3 Network Attached Storage Application ..... 1-12
1-4 Gaming Kiosk Application ..... 1-13
1-5 Network Controller Application ..... 1-14
2-1 Local Memory Map Example ..... 2-2
2-2 Local Access IP Block Revision Register 1 (LAIPBRR1) ..... 2-6
2-3 Local Access IP Block Revision Register 2 (LAIPBRR2) ..... 2-6
2-4 Local Access Window *n* Base Address Registers (LAWBAR0–LAWBAR7) ..... 2-7
2-5 Local Access Window *n* Attributes Registers (LAWAR0–LAWAR7) ..... 2-7
2-6 General Utilities Registers Mapping to Configuration, Control, and Status Memory Block ..... 2-12
2-7 PIC Mapping to Configuration, Control, and Status Memory Block ..... 2-13
2-8 Device-Specific Register Mapping to Configuration, Control, and Status Memory Block ..... 2-14
4-1 Configuration, Control, and Status Registers Base Address Register (CCSRBAR) ..... 4-5
4-2 Alternate Configuration Base Address Register (ALTCBAR) ..... 4-6
4-3 Alternate Configuration Attribute Register (ALTCAR) ..... 4-6
4-4 Boot Page Translation Register (BPTR) ..... 4-7
4-5 Power-On Reset Sequence ..... 4-10
4-6 Clock Subsystem Block Diagram ..... 4-25
4-7 RTC and Core Timer Facilities Clocking Options ..... 4-27
4-8 SD/MMC Card Data Structure ..... 4-29
4-9 Config Address Fields ..... 4-31
4-10 SD/MMC Card Data Structure for Maximum Redundancy ..... 4-35
4-11 eSPI EEPROM Data Structure ..... 4-37
4-12 Config Address Fields ..... 4-39
4-13 External Signal Connection ..... 4-40
4-14 eSPI CS0 Mode Register (SPMODE0) Configuration ..... 4-40
4-15 Read Instruction Timing Diagram (24-bit addressable eSPI memory) ..... 4-41
5-1 e500 Core Complex Block Diagram ..... 5-2
5-2 e500 Core Integration ..... 5-3
6-1 L2 Cache/SRAM Configuration ..... 6-1
6-2 Cache Organization ..... 6-4
6-3 Physical Address Usage for L2 Cache Accesses ..... 6-5
6-4 Physical Address Usage for SRAM Accesses ..... 6-6
6-5 Data Bus Connection of CCB ..... 6-8
6-6 Address Bus Connection of CCB ..... 6-8

# Figures

Figure Number | Title | Page Number

6-7 L2 Control Register (L2CTL) ........ 6-10
6-8 Cache External Write Address Registers (L2CEWAR*n*) ........ 6-13
6-9 Cache External Write Address Registers Extended Address (L2CEWAREA*n*)........ 6-14
6-10 Cache External Write Control Registers (L2CEWCR0–L2CEWCR3) ........ 6-14
6-11 L2 Memory-Mapped SRAM Base Address Registers (L2SRBAR*n*)........ 6-16
6-12 L2 Memory-Mapped SRAM Base Address Registers Extended Address 0–1 (L2SRBAREA*n*) ........ 6-17
6-13 L2 Error Injection Mask High Register (L2ERRINJHI) ........ 6-18
6-14 L2 Error Injection Mask Low Register (L2ERRINJLO)........ 6-18
6-15 L2 Error Injection Mask Control Register (L2ERRINJCTL)........ 6-19
6-16 L2 Error Capture Data High Register (L2CAPTDATAHI)........ 6-20
6-17 L2 Error Capture Data Low Register (L2CAPTDATALO) ........ 6-20
6-18 L2 Error Syndrome Register (L2CAPTECC) ........ 6-20
6-19 L2 Error Detect Register (L2ERRDET) ........ 6-21
6-20 L2 Error Disable Register (L2ERRDIS)........ 6-22
6-21 L2 Error Interrupt Enable Register (L2ERRINTEN) ........ 6-22
6-22 L2 Error Attributes Capture Register (L2ERRATTR)........ 6-23
6-23 L2 Error Address Capture Register (L2ERRADDRL) ........ 6-24
6-24 L2 Error Address Capture Register (L2ERRADDRH)........ 6-25
6-25 L2 Error Control Register (L2ERRCTL)........ 6-25
6-26 L2 Cache Line Replacement Algorithm ........ 6-31
7-1 e500 Coherency Module Block Diagram........ 7-1
7-2 ECM CCB Address Configuration Register (EEBACR)........ 7-3
7-3 ECM CCB Port Configuration Register (EEBPCR)........ 7-4
7-4 ECM IP Block Revision Register 1 (EIPBRR1)........ 7-5
7-5 ECM IP Block Revision Register 2 (EIPBRR2)........ 7-5
7-6 ECM Error Detect Register (EEDR)........ 7-6
7-7 ECM Error Enable Register (EEER) ........ 7-7
7-8 ECM Error Attributes Capture Register (EEATR) ........ 7-7
7-9 ECM Error Low Address Capture Register (EELADR)........ 7-8
7-10 ECM Error High Address Capture Register (EEHADR)........ 7-9
8-1 DDR Memory Controller Simplified Block Diagram........ 8-2
8-2 Chip Select Bounds Registers (CS*n*_BNDS)........ 8-13
8-3 Chip Select Configuration Register (CS*n*_CONFIG)........ 8-13
8-4 Chip Select Configuration Register 2 (CSn_CONFIG_2)........ 8-15
8-5 DDR SDRAM Timing Configuration 3 (TIMING_CFG_3) ........ 8-16
8-6 DDR SDRAM Timing Configuration 0 (TIMING_CFG_0) ........ 8-17
8-7 DDR SDRAM Timing Configuration 1 (TIMING_CFG_1) ........ 8-19
8-8 DDR SDRAM Timing Configuration 2 Register (TIMING_CFG_2)........ 8-21
8-9 DDR SDRAM Control Configuration Register (DDR_SDRAM_CFG)........ 8-23
8-10 DDR SDRAM Control Configuration Register 2 (DDR_SDRAM_CFG_2)........ 8-26

# Figures

Figure Number | Title | Page Number

8-11 DDR SDRAM Mode Configuration Register (DDR_SDRAM_MODE)............................ 8-29
8-12 DDR SDRAM Mode 2 Configuration Register (DDR_SDRAM_MODE_2)...................... 8-29
8-13 DDR SDRAM Mode Control Register (DDR_SDRAM_MD_CNTL)................................ 8-30
8-14 DDR SDRAM Interval Configuration Register (DDR_SDRAM_INTERVAL) ................. 8-33
8-15 DDR SDRAM Data Initialization Configuration Register (DDR_DATA_INIT)................ 8-33
8-16 DDR SDRAM Clock Control Configuration Register (DDR_SDRAM_CLK_CNTL)....... 8-34
8-17 DDR Initialization Address Configuration Register (DDR_INIT_ADDR) ......................... 8-35
8-18 DDR Initialization Extended Address Configuration Register (DDR_INIT_EXT_ADDR)........................................................................................ 8-35
8-19 DDR SDRAM Timing Configuration 4 Register (TIMING_CFG_4)................................ 8-36
8-20 DDR SDRAM Timing Configuration 5 Register (TIMING_CFG_5)................................ 8-37
8-21 DDR ZQ Calibration Control Register (DDR_ZQ_CNTL) ............................................... 8-39
8-22 DDR Write Leveling Control Register (DDR_WRLVL_CNTL) ....................................... 8-40
8-23 DDR Self Refresh Counter Register (DDR_SR_CNTR) .................................................. 8-43
8-24 DDR Register Control Word 1 (DDR_SDRAM_RCW_1) ................................................ 8-44
8-25 DDR Register Control Word 2 (DDR_SDRAM_RCW_2) ................................................ 8-45
8-26 DDR Debug Status Register 1 (DDRDSR_1)..................................................................... 8-46
8-27 DDR Debug Status Register 2 (DDRDSR_2)..................................................................... 8-47
8-28 DDR Control Driver Register 1 (DDRCDR_1) ................................................................. 8-49
8-29 DDR Control Driver Register 2 (DDRCDR_2) ................................................................. 8-50
8-30 DDR IP Block Revision 1 (DDR_IP_REV1) ...................................................................... 8-50
8-31 DDR IP Block Revision 2 (DDR_IP_REV2) ...................................................................... 8-51
8-32 Memory Data Path Error Injection Mask High Register (DATA_ERR_INJECT_HI) ......... 8-51
8-33 Memory Data Path Error Injection Mask Low Register (DATA_ERR_INJECT_LO)......... 8-52
8-34 Memory Data Path Error Injection Mask ECC Register (ERR_INJECT)........................... 8-52
8-35 Memory Data Path Read Capture High Register (CAPTURE_DATA_HI)......................... 8-53
8-36 Memory Data Path Read Capture Low Register (CAPTURE_DATA_LO) ........................ 8-54
8-37 Memory Data Path Read Capture ECC Register (CAPTURE_ECC)................................. 8-54
8-38 Memory Error Detect Register (ERR_DETECT)................................................................ 8-55
8-39 Memory Error Disable Register (ERR_DISABLE)............................................................. 8-56
8-40 Memory Error Interrupt Enable Register (ERR_INT_EN)................................................. 8-57
8-41 Memory Error Attributes Capture Register (CAPTURE_ATTRIBUTES).......................... 8-58
8-42 Memory Error Address Capture Register (CAPTURE_ADDRESS) ................................. 8-58
8-43 Memory Error Extended Address Capture Register (CAPTURE_EXT_ADDRESS).......... 8-59
8-44 Single-Bit ECC Memory Error Management Register (ERR_SBE) .................................. 8-59
8-45 DDR Memory Controller Block Diagram ........................................................................... 8-61
8-46 Typical Dual Data Rate SDRAM Internal Organization..................................................... 8-62
8-47 Typical DDR SDRAM Interface Signals ............................................................................. 8-62
8-48 Example 256-Mbyte DDR SDRAM Configuration With ECC .......................................... 8-63
8-49 DDR SDRAM Burst Read Timing—ACTTORW = 3, $\overline{\text{MCAS}}$ Latency = 2 ........................ 8-74
8-50 DDR SDRAM Single-Beat (Double Word) Write Timing—ACTTOR .............................. 8-75

# Figures

**Figure Number** **Title** **Page Number**

8-51 DDR SDRAM Single-Beat (Double Word) Write Timing—ACTTORW = 3 ... 8-75
8-52 DDR SDRAM Clock Distribution Example for x8 DDR SDRAMs ... 8-76
8-53 DDR SDRAM Mode-Set Command Timing ... 8-77
8-54 Registered DDR SDRAM DIMM Burst Write Timing ... 8-78
8-55 Write Timing Adjustments Example for Write Latency = 1 ... 8-79
8-56 DDR SDRAM Bank Staggered Auto Refresh Timing ... 8-80
8-57 DDR SDRAM Power-Down Mode ... 8-81
8-58 DDR SDRAM Self-Refresh Entry Timing ... 8-82
8-59 DDR SDRAM Self-Refresh Exit Timing ... 8-82
9-1 Interrupt Sources Block Diagram Features ... 9-2
9-2 Pass-Through Mode Example ... 9-5
9-3 Block Revision Register 1 (BRR1) ... 9-19
9-4 Block Revision Register 2 (BRR2) ... 9-19
9-5 Feature Reporting Register (FRR) ... 9-20
9-6 Global Configuration Register (GCR) ... 9-21
9-7 Vendor Identification Register (VIR) ... 9-21
9-8 Processor Core Initialization Register (PIR) ... 9-22
9-9 Interprocessor Interrupt Vector/Priority Register (IPIVPR*n*) ... 9-22
9-10 Spurious Vector Register (SVR) ... 9-23
9-11 Timer Frequency Reporting Registers (TFRR*x*) ... 9-24
9-12 Global Timer Current Count Registers (GTCCR*xn*) ... 9-24
9-13 Global Timer Base Count Register (GTBCR*xn*) ... 9-25
9-14 Global Timer Vector/Priority Register (GTVPR*xn*) ... 9-25
9-15 Global Timer Destination Registers (GTDR*xn*) ... 9-26
9-16 Example Calculation for Cascaded Timers ... 9-27
9-17 Timer Control Registers (TCR*x*) ... 9-27
9-18 External Interrupt Summary Register (ERQSR) ... 9-29
9-19 $\overline{\text{IRQ\_OUT}}$ Summary Register 0 (IRQSR0) ... 9-29
9-20 IRQ_OUT Summary Register 1 (IRQSR1) ... 9-30
9-21 IRQ_OUT Summary Register 2 (IRQSR2) ... 9-31
9-22 Critical Interrupt Summary Register 0 (CISR0) ... 9-31
9-23 Critical Interrupt Summary Register 1 (CISR1) ... 9-32
9-24 Critical Interrupt Summary Register 2 (CISR2) ... 9-32
9-25 Performance Monitor Mask Registers 0 (PM*n*MR0) ... 9-33
9-26 Performance Monitor Mask Registers 1 (PM*n*MR1) ... 9-34
9-27 Performance Monitor Mask Registers 2 (PM*n*MR2) ... 9-34
9-28 Message Registers (MSGRs) ... 9-35
9-29 Message Enable Register (MER) ... 9-35
9-30 Message Status Register (MSR) ... 9-36
9-31 Message Signaled Interrupt Registers (MSIR*n*) ... 9-37
9-32 Shared Message Signaled Interrupt Status Register (MSISR) ... 9-37

# Figures

| Figure Number | Title | Page Number |
|---|---|---|
| 9-33 | Shared Message Signaled Interrupt Index Register (MSIIR) | 9-38 |
| 9-34 | Shared Message Signaled Interrupt Vector/Priority Register (MSIVPRs) | 9-38 |
| 9-35 | Shared Message Signaled Interrupt Destination Registers (MSIDR*n*) | 9-39 |
| 9-36 | Destination Register Summary | 9-40 |
| 9-37 | Vector/Priority Register Summary | 9-41 |
| 9-38 | External Interrupt Vector/Priority Registers (EIVPR0–EIVPR11) | 9-41 |
| 9-39 | External Interrupt Destination Registers (EIDRs) | 9-42 |
| 9-40 | Internal Interrupt Vector/Priority Registers (IIVPRs) | 9-43 |
| 9-41 | Internal Interrupt Destination Registers (IIDRs) | 9-44 |
| 9-42 | Messaging Interrupt Vector/Priority Registers (MIVPR*n*) | 9-45 |
| 9-43 | Messaging Interrupt Destination Registers (MIDR*n*) | 9-46 |
| 9-44 | Per-CPU Register Address Decoding in a Four-Core Device | 9-48 |
| 9-45 | Interprocessor Interrupt Dispatch Registers (IPIDR0–IPIDR3) | 9-48 |
| 9-46 | Processor Core Current Task Priority Registers (CTPR*n*) | 9-49 |
| 9-47 | Processor Core Who Am I Registers (WHOAMI*n*) | 9-50 |
| 9-48 | Processor Core Interrupt Acknowledge Registers (IACK*n*) | 9-51 |
| 9-49 | End of Interrupt Registers (EOI*n*) | 9-51 |
| 9-50 | PIC Interrupt Processing Flow Diagram for Each Core (*n*) | 9-53 |
| 10-1 | SEC Functional Modules | 10-3 |
| 10-2 | Descriptor Format | 10-20 |
| 10-3 | Header Dword | 10-21 |
| 10-4 | Pointer Dword | 10-25 |
| 10-5 | Link Table Entry | 10-26 |
| 10-6 | Descriptors, Link Tables, and Parcels | 10-28 |
| 10-7 | Fetch FIFO Enqueue Counter | 10-36 |
| 10-8 | Descriptor Finished Counter | 10-36 |
| 10-9 | Data Bytes In Counter | 10-37 |
| 10-10 | Data Bytes Out Counter | 10-37 |
| 10-11 | Channel Configuration Register (CCR) | 10-38 |
| 10-12 | Channel Status Register (CSR) | 10-41 |
| 10-13 | Current Descriptor Pointer Register | 10-43 |
| 10-14 | Fetch FIFO Enqueue Register (FFER) | 10-44 |
| 10-15 | Gather/Scatter Link Table Entry Format and Memory Ranges | 10-46 |
| 10-16 | EU Assignment Status Register (EUASR) | 10-50 |
| 10-17 | Interrupt Enable, Interrupt Status, and Interrupt Clear Registers | 10-51 |
| 10-18 | ID Register | 10-54 |
| 10-19 | IP Block Revision Register | 10-54 |
| 10-20 | Master Control Register | 10-55 |
| 10-21 | AESU Mode Register | 10-58 |
| 10-22 | AESU Key Size Register | 10-61 |
| 10-23 | AESU Data Size Register | 10-61 |

# Figures

Figure Number | Title | Page Number

10-24 AESU Reset Control Register ........ 10-62
10-25 AESU Status Register ........ 10-63
10-26 AESU Interrupt Status Register ........ 10-64
10-27 AESU Interrupt Mask Register ........ 10-66
10-28 AESU ICV Size Register ........ 10-67
10-29 AESU End of Message Register ........ 10-68
10-30 AESU CCM Context Registers ........ 10-76
10-31 AESU Key Registers ........ 10-87
10-32 AFEU Mode Register ........ 10-89
10-33 AFEU Key Size Register ........ 10-90
10-34 AFEU Context/Data Size Register ........ 10-91
10-35 AFEU Reset Control Register ........ 10-91
10-36 AFEU Status Register ........ 10-92
10-37 AFEU Interrupt Status Register ........ 10-93
10-38 AFEU Interrupt Mask Register ........ 10-94
10-39 AFEU End of Message Register ........ 10-96
10-40 CRCU Mode Register ........ 10-99
10-41 CRCU Key Size Register ........ 10-100
10-42 CRCU Data Size Register ........ 10-100
10-43 CRCU Reset Control Register ........ 10-100
10-44 CRCU Control Register ........ 10-101
10-45 CRCU Status Register ........ 10-101
10-46 CRCU Interrupt Status Register ........ 10-103
10-47 CRCU Interrupt Mask Register ........ 10-104
10-48 CRCU ICV Size Register ........ 10-106
10-49 CRCU End of Message Register ........ 10-106
10-50 CRCU Context Register (Write) ........ 10-107
10-51 CRCU Context Register, Upper Half (Read in Default Mode) ........ 10-107
10-52 CRCU Context Register, Upper Half (Read in DOS Mode) ........ 10-107
10-53 CRCU Context Register, Upper Half (Read in DOC Mode) ........ 10-107
10-54 CRCU Context Register, Upper Half (Read in DOC and DOS Modes) ........ 10-108
10-55 CRCU Key Register ........ 10-108
10-56 DEU Mode Register ........ 10-109
10-57 DEU Key Size Register ........ 10-110
10-58 DEU Data Size Register ........ 10-111
10-59 DEU Reset Control Register ........ 10-111
10-60 DEU Status Register ........ 10-112
10-61 DEU Interrupt Status Register ........ 10-113
10-62 DEU Interrupt Mask Register ........ 10-115
10-63 DEU End of Message Register ........ 10-116
10-64 KEU Mode Register ........ 10-118

# Figures

Figure Number | Title | Page Number

10-65 KEU Key Size Register ........ 10-119
10-66 KEU Data Size Register ........ 10-120
10-67 KEU Reset Control Register ........ 10-121
10-68 KEU Status Register ........ 10-122
10-69 KEU Interrupt Status Register ........ 10-124
10-70 KEU Interrupt Mask Register ........ 10-125
10-71 KEU Data Out Register (f9 MAC) ........ 10-127
10-72 KEU End of Message Register ........ 10-127
10-73 KEU IV_1 Register ........ 10-128
10-74 KEU IV_2 Register (FRESH) ........ 10-129
10-75 KEU Key Data Register_1 (CK-high) ........ 10-130
10-76 KEU Key Data Register_1 (CK-high) ........ 10-130
10-77 KEU Key Data Register_2 (CK-Low) ........ 10-130
10-78 KEU Key Data Register_2 (CK-Low) ........ 10-130
10-79 KEU Key Data Register_3 (IK-high) ........ 10-131
10-80 KEU Key Data Register_3 (IK-high) ........ 10-131
10-81 KEU Key Data Register_4 (IK-low) ........ 10-131
10-82 KEU Key Data Register_4 (IK-low) ........ 10-131
10-83 MDEU Mode Register in Old Configuration ........ 10-133
10-84 MDEU Mode Register in New Configuration ........ 10-134
10-85 MDEU Key Size Register ........ 10-136
10-86 MDEU Data Size Register ........ 10-137
10-87 MDEU Reset Control Register ........ 10-137
10-88 MDEU Status Register ........ 10-138
10-89 MDEU Interrupt Status Register ........ 10-140
10-90 MDEU Interrupt Mask Register ........ 10-141
10-91 MDEU ICV Size Register ........ 10-142
10-92 MDEU End of Message Register ........ 10-143
10-93 MDEU Context Registers ........ 10-144
10-94 PKEU Mode Register ........ 10-147
10-95 PKEU Key Size Register ........ 10-147
10-96 PKEU AB Size Register ........ 10-149
10-97 PKEU Data Size Register ........ 10-149
10-98 PKEU Reset Control Register ........ 10-149
10-99 PKEU Status Register ........ 10-150
10-100 PKEU Interrupt Status Register ........ 10-152
10-101 PKEU Interrupt Mask Register ........ 10-153
10-102 PKEU End of Message Register ........ 10-154
10-103 RNGU Mode Register ........ 10-156
10-104 RNGU Data Size Register ........ 10-156
10-105 RNGU Reset Control Register ........ 10-156

# Figures

| Figure Number | Title | Page Number |
|---|---|---|
| 10-106 | RNGU Status Register | 10-157 |
| 10-107 | RNGU Interrupt Status Register | 10-159 |
| 10-108 | RNGU Interrupt Mask Register | 10-160 |
| 10-109 | RNGU End of Message Register | 10-160 |
| 11-1 | $I^2C$ Block Diagram | 11-1 |
| 11-2 | $I^2C$ Address Register (I2CADR) | 11-6 |
| 11-3 | $I^2C$ Frequency Divider Register (I2CFDR) | 11-6 |
| 11-4 | $I^2C$ Control Register (I2CCR) | 11-7 |
| 11-5 | $I^2C$ Status Register (I2CSR) | 11-9 |
| 11-6 | $I^2C$ Data Register (I2CDR) | 11-10 |
| 11-7 | $I^2C$ Digital Filter Sampling Rate Register (I2CDFSRR) | 11-11 |
| 11-8 | $I^2C$ Interface Transaction Protocol | 11-12 |
| 11-9 | EEPROM Data Format for One Register Preload Command | 11-19 |
| 11-10 | EEPROM Contents | 11-20 |
| 11-11 | Example $I^2C$ Interrupt Service Routine Flowchart | 11-25 |
| 12-1 | UART Block Diagram | 12-2 |
| 12-2 | Receiver Buffer Registers (URBR*n*) | 12-5 |
| 12-3 | Transmitter Holding Registers (UTHR*n*) | 12-6 |
| 12-4 | Divisor Most Significant Byte Registers (UDMB0, UDMB1) | 12-6 |
| 12-5 | Divisor Least Significant Byte Registers (UDLB*n*) | 12-7 |
| 12-6 | Interrupt Enable Register (UIER) | 12-8 |
| 12-7 | Interrupt ID Registers (UIIR) | 12-9 |
| 12-8 | FIFO Control Registers (UFCR*n*) | 12-10 |
| 12-9 | Alternate Function Register (UAFR) | 12-11 |
| 12-10 | Line Control Register (ULCR) | 12-12 |
| 12-11 | Modem Control Register (UMCR) | 12-14 |
| 12-12 | Line Status Register (ULSR) | 12-15 |
| 12-13 | Modem Status Register (UMSR) | 12-16 |
| 12-14 | Scratch Register (USCR) | 12-17 |
| 12-15 | DMA Status Register (UDSR) | 12-17 |
| 12-16 | UART Bus Interface Transaction Protocol Example | 12-19 |
| 13-1 | Enhanced Local Bus Controller Block Diagram | 13-1 |
| 13-2 | Base Registers (BR*n*) | 13-11 |
| 13-3 | Option Registers (OR*n*) in GPCM Mode | 13-14 |
| 13-4 | Option Registers (OR*n*) in FCM Mode | 13-16 |
| 13-5 | Option Registers (OR*n*) in UPM Mode | 13-19 |
| 13-6 | UPM Memory Address Register (MAR) | 13-20 |
| 13-7 | UPM Mode Registers (M*x*MR) | 13-21 |
| 13-8 | Memory Refresh Timer Prescaler Register (MRTPR) | 13-23 |
| 13-9 | UPM Data Register in UPM Mode (MDR) | 13-24 |
| 13-10 | FCM Data Register in FCM Mode (MDR) | 13-24 |

# Figures

Figure Number | Title | Page Number

13-11 Special Operation Initiation Register (LSOR) ........ 13-25
13-12 UPM Refresh Timer (LURT) ........ 13-25
13-13 Transfer Error Status Register (LTESR) ........ 13-26
13-14 Transfer Error Check Disable Register (LTEDR) ........ 13-28
13-15 Transfer Error Interrupt Enable Register (LTEIR) ........ 13-29
13-16 Transfer Error Attributes Register (LTEATR ........ 13-30
13-17 Transfer Error Address Register (LTEAR) ........ 13-31
13-18 Transfer Error ECC Register (LTECCR) ........ 13-32
13-19 Local Bus Configuration Register ........ 13-32
13-20 Clock Ratio Register (LCRR) ........ 13-34
13-21 Flash Mode Register ........ 13-35
13-22 Flash Instruction Register ........ 13-37
13-23 Flash Command Register ........ 13-38
13-24 Flash Block Address Register ........ 13-39
13-25 Flash Page Address Register, Small Page Device (ORx[PGS] = 0) ........ 13-39
13-26 Flash Page Address Register, Large Page Device (ORx[PGS] = 1) ........ 13-39
13-27 Flash Byte Count Register ........ 13-41
13-28 Flash ECC Blockn Register (FECC0–FECC3) ........ 13-41
13-29 Basic Operation of Memory Controllers in the eLBC ........ 13-43
13-30 Example of 8-Bit GPCM Writing 32 Bytes to Address 0x5420 (LCRR[PBYP] = 0) ........ 13-45
13-31 Basic eLBC Bus Cycle with LALE, TA, and $\overline{\text{LCSn}}$ ........ 13-46
13-32 eLBC Bus Cycles in PLL Mode (GPCM and UPM only) ........ 13-48
13-33 eLBC Bus Cycles in PLL-bypassed Mode (GPCM and UPM only) ........ 13-48
13-34 Enhanced Local Bus to GPCM Device Interface ........ 13-49
13-35 GPCM Basic Read Timing (XACS = 0, ACS = 1x, TRLX = 0) ........ 13-49
13-36 GPCM General Read Timing Parameters ........ 13-50
13-37 GPCM General Write Timing Parameters ........ 13-51
13-38 GPCM Basic Write Timing
(XACS = 0, ACS = 00, CSNT = 1, SCY = 1, TRLX = 0) ........ 13-53
13-39 GPCM Relaxed Timing Back-to-Back Reads
(XACS = 0, ACS = 1x, SCY = 1, CSNT = 0, TRLX = 1, EHTR = 0) ........ 13-54
13-40 GPCM Relaxed Timing Back-to-Back Writes
(XACS = 0, ACS = 1x, SCY = 0, CSNT = 0, TRLX = 1) ........ 13-55
13-41 GPCM Relaxed Timing Write
(XACS = 0, ACS = 10, SCY = 0, CSNT = 1, TRLX = 1) ........ 13-55
13-42 GPCM Relaxed Timing Write
(XACS = 0, ACS = 00, SCY = 1, CSNT = 1, TRLX = 1) ........ 13-56
13-43 GPCM Read Followed by Read (TRLX = 0, EHTR = 0, Fastest Timing) ........ 13-57
13-44 GPCM Read Followed by Write
(TRLX = 0, EHTR = 1, One-Cycle Extended Hold Time on Reads) ........ 13-57
13-45 External Termination of GPCM Access ........ 13-58

# Figures

Figure Number | Title | Page Number

13-46 Local Bus to 8-Bit FCM Device Interface ... 13-60
13-47 FCM Basic Page Read Timing
(PGS = 1, CSCT = 0, CST = 0, CHT = 1, RST = 1, SCY = 0, TRLX = 0, EHTR = 1) 13-60
13-48 FCM Buffer RAM Memory Map for Small-Page (512-byte page) NAND Flash Devices 13-62
13-49 FCM Buffer RAM Memory Map for Large-Page (2-Kbyte page) NAND Flash Devices . 13-63
13-50 FCM ECC Calculation ... 13-64
13-51 ECC Layout for LBCR[EPAR] = 0 (~ represents logical negation) ... 13-64
13-52 ECC Placement in NAND Flash Spare Regions in Relation to FMR[ECCM] ... 13-65
13-53 FCM Instruction Sequencer Mechanism ... 13-65
13-54 Timing of FCM Command/Address and Write Data Cycles
(for TRLX = 0, CHT = 0, CST = 1, SCY = 1) ... 13-68
13-55 Example of FCM Command and Address Timing with Minimum Delay Parameters
(for TRLX = 0, CHT = 0, CST = 0, SCY = 0) ... 13-69
13-56 Example of FCM Command and Address Timing with Relaxed Parameters
(for TRLX = 1, CHT = 0, CST = 1, SCY = 2) ... 13-70
13-57 FCM Delay Prior to Sampling LFR$\overline{\text{B}}$ State ... 13-70
13-58 FCM Read Data Timing
(for TRLX = 0, RST = 0, SCY = 1) ... 13-71
13-59 FCM Read Data Timing with Extended Hold Time
(for TRLX = 0, EHTR = 1, RST = 1, SCY = 1) ... 13-72
13-60 FCM Buffer RAM Memory Map During Boot Loading ... 13-73
13-61 User-Programmable Machine Functional Block Diagram ... 13-75
13-62 RAM Array Indexing ... 13-76
13-63 Memory Refresh Timer Request Block Diagram ... 13-77
13-64 UPM Clock Scheme ... 13-80
13-65 RAM Array and Signal Generation ... 13-80
13-66 RAM Word Fields ... 13-81
13-67 $\overline{\text{LCSn}}$ Signal Selection ... 13-84
13-68 LBS Signal Selection ... 13-85
13-69 UPM Read Access Data Sampling ... 13-88
13-70 Effect of LUPWAIT Signal ... 13-89
13-71 Multiplexed Address/Data Bus for 32-Bit Addressing ... 13-91
13-72 Local Bus Peripheral Hierarchy for High Bus Speeds ... 13-91
13-73 GPCM Address Timings ... 13-92
13-74 GPCM Data Timings ... 13-92
13-75 Interface to Different Port-Size Devices ... 13-94
13-76 Single-Beat Read Access to FPM DRAM ... 13-100
13-77 Single-Beat Write Access to FPM DRAM ... 13-101
13-78 Burst Read Access to FPM DRAM Using LOOP (Two Beats Shown) ... 13-102
13-79 Refresh Cycle (CBR) to FPM DRAM ... 13-103
13-80 Exception Cycle ... 13-104

# Figures

| Figure Number | Title | Page Number |
|---|---|---|
| 13-81 | Interface to ZBT SRAM | 13-105 |
| 14-1 | eTSEC Block Diagram | 14-2 |
| 14-2 | TSEC_ID Register | 14-26 |
| 14-3 | TSEC_ID2 Register | 14-27 |
| 14-4 | IEVENT Register Definition | 14-28 |
| 14-5 | IMASK Register Definition | 14-32 |
| 14-6 | EDIS Register Definition | 14-33 |
| 14-7 | ECNTRL Register Definition | 14-35 |
| 14-8 | PTV Register Definition | 14-38 |
| 14-9 | DMACTRL Register | 14-38 |
| 14-10 | TBIPA Register Definition | 14-40 |
| 14-11 | TCTRL Register Definition | 14-40 |
| 14-12 | TSTAT Register Definition | 14-42 |
| 14-13 | DFVLAN Register Definition | 14-46 |
| 14-14 | TXIC Register Definition | 14-47 |
| 14-15 | TQUEUE Register Definition | 14-48 |
| 14-16 | TR03WT Register Definition | 14-49 |
| 14-17 | TR47WT Register Definition | 14-49 |
| 14-18 | TBDBPH Register Definition | 14-50 |
| 14-19 | TBPTR0–TBPTR7 Register Definition | 14-51 |
| 14-20 | TBASEH Register Definition | 14-51 |
| 14-21 | TBASE Register Definition | 14-52 |
| 14-22 | TMR_TXTS*n*_ID Register Definition | 14-52 |
| 14-23 | TMR_TXTS*n*_H/L Register Definition | 14-53 |
| 14-24 | RCTRL Register Definition | 14-54 |
| 14-25 | RSTAT Register Definition | 14-57 |
| 14-26 | RXIC Register Definition | 14-59 |
| 14-27 | RQUEUE Register Definition | 14-60 |
| 14-28 | RBIFX Register Definition | 14-62 |
| 14-29 | Receive Queue Filer Table Address Register Definition | 14-63 |
| 14-30 | Receive Queue Filer Table Control Register Definition | 14-64 |
| 14-31 | Receive Queue Filer Table Property IDs 0, 2–15 Register Definition | 14-65 |
| 14-32 | Receive Queue Filer Table Property ID1 Register Definition | 14-65 |
| 14-33 | MRBLR Register Definition | 14-68 |
| 14-34 | RBDBPH Register Definition | 14-69 |
| 14-35 | RBPTR0–RBPTR7 Register Definition | 14-69 |
| 14-36 | RBASEH Register Definition | 14-70 |
| 14-37 | RBASE Register Definition | 14-70 |
| 14-38 | TMR_RXTS_H/L Register Definition | 14-71 |
| 14-39 | MACCFG1 Register Definition | 14-74 |
| 14-40 | MACCFG2 Register Definition | 14-76 |

# Figures

Figure Number | Title | Page Number

14-41 IPGIFG Register Definition ... 14-78
14-42 Half-Duplex Register Definition ... 14-79
14-43 Maximum Frame Length Register Definition ... 14-80
14-44 MII Management Configuration Register Definition ... 14-80
14-45 MIIMCOM Register Definition ... 14-81
14-46 MIIMADD Register Definition ... 14-82
14-47 MII Mgmt Control Register Definition ... 14-82
14-48 MIIMSTAT Register Definition ... 14-83
14-49 MII Mgmt Indicator Register Definition ... 14-83
14-50 Interface Status Register Definition ... 14-84
14-51 MAC Station Address Part 1 Register Definition ... 14-85
14-52 MAC Station Address Part 2 Register Definition ... 14-85
14-53 MAC Exact Match Address *n* Part 1 Register Definition ... 14-86
14-54 MAC Exact Match Address *x* Part 2 Register Definition ... 14-86
14-55 Transmit and Receive 64-Byte Frame Register Definition ... 14-88
14-56 Transmit and Receive 65- to 127-Byte Frame Register Definition ... 14-88
14-57 Transmit and Received 128- to 255-Byte Frame Register Definition ... 14-89
14-58 Transmit and Received 256- to 511-Byte Frame Register Definition ... 14-89
14-59 Transmit and Received 512- to 1023-Byte Frame Register Definition ... 14-90
14-60 Transmit and Received 1024- to 1518-Byte Frame Register Definition ... 14-90
14-61 Transmit and Received 1519- to 1522-Byte VLAN Frame Register Definition ... 14-91
14-62 Receive Byte Counter Register Definition ... 14-91
14-63 Receive Packet Counter Register Definition ... 14-92
14-64 Receive FCS Error Counter Register Definition ... 14-92
14-65 Receive Multicast Packet Counter Register Definition ... 14-93
14-66 Receive Broadcast Packet Counter Register Definition ... 14-93
14-67 Receive Control Frame Packet Counter Register Definition ... 14-94
14-68 Receive Pause Frame Packet Counter Register Definition ... 14-94
14-69 Receive Unknown OPCode Packet Counter Register Definition ... 14-95
14-70 Receive Alignment Error Counter Register Definition ... 14-95
14-71 Receive Frame Length Error Counter Register Definition ... 14-96
14-72 Receive Code Error Counter Register Definition ... 14-96
14-73 Receive Carrier Sense Error Counter Register Definition ... 14-97
14-74 Receive Undersize Packet Counter Register Definition ... 14-97
14-75 Receive Oversize Packet Counter Register Definition ... 14-98
14-76 Receive Fragments Counter Register Definition ... 14-98
14-77 Receive Jabber Counter Register Definition ... 14-99
14-78 Receive Dropped Packet Counter Register Definition ... 14-99
14-79 Transmit Byte Counter Register Definition ... 14-100
14-80 Transmit Packet Counter Register Definition ... 14-100
14-81 Transmit Multicast Packet Counter Register Definition ... 14-101

# Figures

| Figure Number | Title | Page Number |
|---|---|---|
| 14-82 | Transmit Broadcast Packet Counter Register Definition | 14-101 |
| 14-83 | Transmit Pause Control Frame Counter Register Definition | 14-102 |
| 14-84 | Transmit Deferral Packet Counter Register Definition | 14-102 |
| 14-85 | Transmit Excessive Deferral Packet Counter Register Definition | 14-103 |
| 14-86 | Transmit Single Collision Packet Counter Register Definition | 14-103 |
| 14-87 | Transmit Multiple Collision Packet Counter Register Definition | 14-104 |
| 14-88 | Transmit Late Collision Packet Counter Register Definition | 14-104 |
| 14-89 | Transmit Excessive Collision Packet Counter Register Definition | 14-105 |
| 14-90 | Transmit Total Collision Counter Register Definition | 14-105 |
| 14-91 | Transmit Drop Frame Counter Register Definition | 14-106 |
| 14-92 | Transmit Jabber Frame Counter Register Definition | 14-106 |
| 14-93 | Transmit FCS Error Counter Register Definition | 14-107 |
| 14-94 | Transmit Control Frame Counter Register Definition | 14-107 |
| 14-95 | Transmit Oversized Frame Counter Register Definition | 14-108 |
| 14-96 | Transmit Undersize Frame Counter Register Definition | 14-108 |
| 14-97 | Transmit Fragment Counter Register Definition | 14-109 |
| 14-98 | Carry Register 1 (CAR1) Register Definition | 14-109 |
| 14-99 | Carry Register 2 (CAR2) Register Definition | 14-111 |
| 14-100 | Carry Mask Register 1 (CAM1) Register Definition | 14-112 |
| 14-101 | Carry Mask Register 2 (CAM2) Register Definition | 14-113 |
| 14-102 | Receive Filer Rejected Packet Counter Register Definition | 14-114 |
| 14-103 | IGADDR*n* Register Definition | 14-115 |
| 14-104 | GADDR*n* Register Definition | 14-116 |
| 14-105 | FIFOCFG Register Definition | 14-117 |
| 14-106 | ATTR Register Definition | 14-118 |
| 14-107 | ATTRELI Register Definition | 14-119 |
| 14-108 | RQPRM Register Definition | 14-120 |
| 14-109 | RFBPTR0–RFBPTR7 Register Definition | 14-121 |
| 14-110 | TMR_CTRL Register Definition | 14-122 |
| 14-111 | TMR_TEVENT Register Definition | 14-124 |
| 14-112 | TMR_PEVENT Register Definition | 14-127 |
| 14-113 | TMR_PEMASK Register Definition | 14-127 |
| 14-114 | TMR_CNT_H Register Definition | 14-129 |
| 14-115 | TMR_ADD Register Definition | 14-129 |
| 14-116 | TMR_ACC Register Definition | 14-130 |
| 14-117 | TMR_PRSC Register Definition | 14-130 |
| 14-118 | TMROFF_H/L Register Definition | 14-131 |
| 14-119 | TMR_ALARM1-2_H/L Register Definition | 14-131 |
| 14-120 | TMR_FIPER*n* Register Definition | 14-133 |
| 14-121 | TMR_ETTS1-2_H/L Register Definition | 14-133 |
| 14-122 | Control Register Definition | 14-136 |

# Figures

| Figure Number | Title | Page Number |
|---|---|---|
| 14-123 | Status Register Definition | 14-137 |
| 14-124 | AN Advertisement Register Definition | 14-138 |
| 14-125 | AN Link Partner Base Page Ability Register Definition | 14-140 |
| 14-126 | AN Expansion Register Definition | 14-141 |
| 14-127 | AN Next Page Transmit Register Definition | 14-141 |
| 14-128 | AN Link Partner Ability Next Page Register Definition | 14-142 |
| 14-129 | Extended Status Register Definition | 14-143 |
| 14-130 | Jitter Diagnostics Register Definition | 14-144 |
| 14-131 | TBI Control Register Definition | 14-145 |
| 14-132 | eTSEC-MII Connection | 14-146 |
| 14-133 | eTSEC-RMII Connection | 14-147 |
| 14-134 | eTSEC-GMII Connection | 14-148 |
| 14-135 | eTSEC-RGMII Connection | 14-149 |
| 14-136 | eTSEC-TBI Connection | 14-150 |
| 14-137 | eTSEC-RTBI Connection | 14-151 |
| 14-138 | eTSEC-FIFO (8-Bit) Connection | 14-158 |
| 14-139 | 8-Bit GMII-Style Packet FIFO Timing | 14-158 |
| 14-140 | 8-Bit Encoded Packet FIFO Timing | 14-159 |
| 14-141 | Definition of Custom Preamble Sequence | 14-165 |
| 14-142 | Definition of Received Preamble Sequence | 14-166 |
| 14-143 | Ethernet Address Recognition Flowchart | 14-167 |
| 14-144 | Sample C Code for Computing eTSEC Hash Table Indices | 14-169 |
| 14-145 | Location of Frame Control Blocks for TOE Parameters | 14-177 |
| 14-146 | Transmit Frame Control Block | 14-178 |
| 14-147 | Receive Frame Control Block | 14-179 |
| 14-148 | Structure of the Receive Queue Filer Table | 14-184 |
| 14-149 | 1588 Timer Design Partition | 14-195 |
| 14-150 | Ethernet Sampling Points for 1588 | 14-195 |
| 14-151 | PTP Packet Format | 14-197 |
| 14-152 | Buffer Format for Transmit Time-Stamp Insertion | 14-198 |
| 14-153 | Transmit Frame Control Block | 14-199 |
| 14-154 | Example of eTSEC Memory Structure for BDs | 14-201 |
| 14-155 | Buffer Descriptor Ring | 14-202 |
| 14-156 | Transmit Buffer Descriptor | 14-202 |
| 14-157 | Mapping of TxBDs to a C Data Structure | 14-203 |
| 14-158 | Receive Buffer Descriptor | 14-205 |
| 14-159 | Mapping of RxBDs to a C Data Structure | 14-206 |
| 15-1 | DMA Block Diagram | 15-1 |
| 15-2 | DMA Operational Flow Chart | 15-4 |
| 15-3 | DMA Signal Summary | 15-4 |
| 15-4 | DMA Mode Registers (MR*n*) | 15-9 |

# Figures

| Figure Number | Title | Page Number |
|---|---|---|
| 15-5 | Status Registers (SR*n*) | 15-11 |
| 15-6 | Basic Chaining Mode Flow Chart | 15-13 |
| 15-7 | Extended Current Link Descriptor Address Registers (ECLNDARn) | 15-14 |
| 15-8 | Current Link Descriptor Address Registers (CLNDAR*n*) | 15-14 |
| 15-9 | Source Attributes Registers (SATR*n*) | 15-15 |
| 15-10 | Source Address Registers (SAR*n*) | 15-16 |
| 15-11 | Destination Attributes Registers (DATR*n*) | 15-16 |
| 15-12 | Destination Address Registers (DAR*n*) | 15-17 |
| 15-13 | Byte Count Registers (BCR*n*) | 15-18 |
| 15-14 | Next Link Descriptor Address Registers (NLNDAR*n*) | 15-18 |
| 15-15 | Extended Next Link Descriptor Address Registers (ENLNDAR*n*) | 15-19 |
| 15-16 | Extended Current List Descriptor Address Registers (ECLSDAR*n*) | 15-20 |
| 15-17 | Current List Descriptor Address Registers (CLSDAR*n*) | 15-20 |
| 15-18 | Extended Next List Descriptor Address Registers (ENLSDAR*n*) | 15-21 |
| 15-19 | Next List Descriptor Address Registers (NLSDAR*n*) | 15-21 |
| 15-20 | Source Stride Registers (SSR*n*) | 15-22 |
| 15-21 | Destination Stride Registers (DSR*n*) | 15-22 |
| 15-22 | DMA General Status Register (DGSR) | 15-23 |
| 15-23 | External Control Interface Timing | 15-30 |
| 15-24 | Stride Size and Stride Distance | 15-32 |
| 15-25 | DMA Transaction Flow with DMA Descriptors | 15-35 |
| 15-26 | List Descriptor Format | 15-36 |
| 15-27 | Link Descriptor Format | 15-36 |
| 15-28 | DMA Data Paths | 15-38 |
| 16-1 | PCI Controller Block Diagram | 16-2 |
| 16-2 | PCI Interface External Signals | 16-6 |
| 16-3 | PCI CFG_ADDR Register | 16-14 |
| 16-4 | PCI CFG_DATA Register | 16-15 |
| 16-5 | PCI INT_ACK Register | 16-15 |
| 16-6 | PCI Outbound Translation Address Registers (POTAR*n*) | 16-16 |
| 16-7 | PCI Outbound Translation Extended Address Registers (POTEAR*n*) | 16-16 |
| 16-8 | PCI Outbound Window Base Address Registers (POWBAR*n*) | 16-17 |
| 16-9 | PCI Outbound Window 0 (Default) Attributes Register (POWAR0) | 16-18 |
| 16-10 | PCI Outbound Window 1–4 Attributes Registers (POWAR1–POWAR4) | 16-18 |
| 16-11 | PCI Inbound Translation Address Registers (PITAR*n*) | 16-20 |
| 16-12 | PCI Inbound Window Base Address Registers | 16-20 |
| 16-13 | PCI Inbound Window Base Extended Address Registers (PIWBEAR*n*) | 16-21 |
| 16-14 | PCI Inbound Window Attributes Registers | 16-21 |
| 16-15 | PCI Error Detect Register (ERR_DR) | 16-24 |
| 16-16 | PCI Error Capture Disable Register (ERR_CAP_DR) | 16-25 |
| 16-17 | PCI Error Enable Register (ERR_EN) | 16-26 |

# Figures

| Figure Number | Title | Page Number |
|---|---|---|
| 16-18 | PCI Error Attributes Capture Register (ERR_ATTRIB) | 16-27 |
| 16-19 | PCI Error Address Capture Register (ERR_ADDR) | 16-28 |
| 16-20 | PCI Error Extended Address Capture Register (ERR_EXT_ADDR) | 16-28 |
| 16-21 | PCI Error Data Low Capture Register (ERR_DL) | 16-29 |
| 16-22 | PCI Error Data High Capture Register (ERR_DH) | 16-29 |
| 16-23 | PCI Gasket Timer Register (GAS_TIMR) | 16-29 |
| 16-24 | Common PCI Configuration Header | 16-30 |
| 16-25 | PCI Vendor ID Register | 16-30 |
| 16-26 | PCI Device ID Register | 16-31 |
| 16-27 | PCI Bus Command Register | 16-31 |
| 16-28 | PCI Bus Status Register | 16-33 |
| 16-29 | PCI Revision ID Register | 16-34 |
| 16-30 | PCI Bus Programming Interface Register | 16-34 |
| 16-31 | PCI Subclass Code Register | 16-35 |
| 16-32 | PCI Bus Base Class Code Register | 16-35 |
| 16-33 | PCI Bus Cache Line Size Register | 16-35 |
| 16-34 | PCI Bus Latency Timer Register | 16-36 |
| 16-35 | PCI Configuration and Status Register Base Address Register (PCSRBAR) | 16-37 |
| 16-36 | 32-Bit Memory Base Address Register | 16-37 |
| 16-37 | 64-Bit Low Memory Base Address Register | 16-37 |
| 16-38 | 64-Bit High Memory Base Address Register | 16-38 |
| 16-39 | PCI Subsystem Vendor ID Register | 16-38 |
| 16-40 | PCI Subsystem ID Register | 16-39 |
| 16-41 | PCI Bus Capabilities Pointer Register | 16-39 |
| 16-42 | PCI Bus Interrupt Line Register | 16-39 |
| 16-43 | PCI Bus Interrupt Pin Register | 16-40 |
| 16-44 | PCI Bus Minimum Grant Register (MIN_GNT) | 16-40 |
| 16-45 | PCI Bus Maximum Latency Register (MAX_LAT) | 16-41 |
| 16-46 | PCI Bus Function Register | 16-41 |
| 16-47 | PCI Bus Arbiter Configuration Register | 16-42 |
| 16-48 | PCI Arbitration Example | 16-44 |
| 16-49 | PCI Single-Beat Read Transaction | 16-51 |
| 16-50 | PCI Burst Read Transaction | 16-51 |
| 16-51 | PCI Single-Beat Write Transaction | 16-52 |
| 16-52 | PCI Burst Write Transaction | 16-52 |
| 16-53 | PCI Target-Initiated Terminations | 16-55 |
| 16-54 | DAC Single-Beat Read Example | 16-57 |
| 16-55 | DAC Burst Read Example | 16-57 |
| 16-56 | DAC Single-Beat Write Example | 16-58 |
| 16-57 | DAC Burst Write Example | 16-58 |
| 16-58 | Standard PCI Configuration Header | 16-59 |

# Figures

**Figure Number** **Title** **Page Number**

16-59 PCI Type 0 Configuration Translation ..... 16-62
16-60 PCI Parity Operation ..... 16-65
16-61 Address Invariant Byte Ordering—4 bytes Outbound ..... 16-69
16-62 Address Invariant Byte Ordering—4 bytes Inbound ..... 16-69
16-63 Address Invariant Byte Ordering—8 bytes Outbound ..... 16-70
16-64 Address Invariant Byte Ordering—2 bytes Inbound ..... 16-70
16-65 CFG_DATA Byte Ordering ..... 16-70
17-1 PCI Express Controller Block Diagram ..... 17-2
17-2 PCI Express Configuration Address Register (PEX_CONFIG_ADDR) ..... 17-10
17-3 PCI Express Configuration Data Register (PEX_CONFIG_DATA) ..... 17-11
17-4 PCI Express Outbound Completion Timeout Register (PEX_OTB_CPL_TOR) ..... 17-11
17-5 PCI Express Configuration Retry Timeout Register (PEX_CONF_RTY_TOR) ..... 17-12
17-6 PCI Express Configuration Register (PEX_CONFIG) ..... 17-12
17-7 PCI Express PME and Message Detect Register (PEX_PME_MES_DR) ..... 17-13
17-8 PCI Express PME and Message Disable Register (PEX_PME_MES_DISR) ..... 17-15
17-9 PCI Express PME and Message Interrupt Enable Register (PEX_PME_MES_IER) ..... 17-17
17-10 PCI Express Power Management Command Register (PEX_PMCR) ..... 17-18
17-11 IP Block Revision Register 1 ..... 17-19
17-12 IP Block Revision Register 2 ..... 17-19
17-13 RC Outbound Transaction Flow ..... 17-20
17-14 PCI Express Outbound Translation Address Registers (PEXOTAR*n*) ..... 17-21
17-15 PCI Express Outbound Translation Extended Address Registers (PEXOTEAR*n*) ..... 17-21
17-16 PCI Express Outbound Window Base Address Registers (PEXOWBAR*n*) ..... 17-22
17-17 PCI Express Outbound Window Attributes Register 0 (PEXOWAR0) ..... 17-22
17-18 PCI Express Outbound Window Attributes Registers 1–4 (PEXOWAR*n*) ..... 17-23
17-19 RC Inbound Transaction Flow ..... 17-26
17-20 PCI Express Inbound Translation Address Registers (PEXITAR*n*) ..... 17-26
17-21 PCI Express Inbound Window Base Address Registers (PEXIWBAR*n*) ..... 17-27
17-22 PCI Express Inbound Window Base Extended Address Registers (PEXIWBEAR*n*) ..... 17-27
17-23 PCI Express Inbound Window Attributes Registers (PEXIWAR*n*) ..... 17-28
17-24 PCI Express Error Detect Register (PEX_ERR_DR) ..... 17-30
17-25 PCI Express Error Interrupt Enable Register (PEX_ERR_EN) ..... 17-32
17-26 PCI Express Error Disable Register (PEX_ERR_DISR) ..... 17-34
17-27 PCI Express Error Capture Status Register (PEX_ERR_CAP_STAT) ..... 17-36
17-28 PCI Express Error Capture Register 0 (PEX_ERR_CAP_R0)
Internal Source, Outbound Transaction ..... 17-37
17-29 PCI Express Error Capture Register 0 (PEX_ERR_CAP_R0)
External Source, Inbound Transaction ..... 17-37
17-30 PCI Express Error Capture Register 1 (PEX_ERR_CAP_R1)
Internal Source, Outbound Transaction ..... 17-38

# Figures

Figure Number | Title | Page Number

17-31 PCI Express Error Capture Register 1 (PEX_ERR_CAP_R1) External Source, Inbound Transaction ........ 17-39
17-32 PCI Express Error Capture Register 2 (PEX_ERR_CAP_R2) Internal Source, Outbound Transaction ........ 17-40
17-33 PCI Express Error Capture Register 2 (PEX_ERR_CAP_R2) External Source, Inbound Transaction ........ 17-41
17-34 PCI Express Error Capture Register 3 (PEX_ERR_CAP_R3) Internal Source, Outbound Transaction ........ 17-42
17-35 PCI Express Error Capture Register 3 (PEX_ERR_CAP_R3) External Source, Inbound Transaction ........ 17-42
17-36 PCI Express PCI-Compatible Configuration Header Common Registers ........ 17-45
17-37 PCI Express Vendor ID Register ........ 17-45
17-38 PCI Express Device ID Register ........ 17-46
17-39 PCI Express Command Register ........ 17-46
17-40 PCI Express Status Register ........ 17-47
17-41 PCI Express Revision ID Register ........ 17-48
17-42 PCI Express Class Code Register ........ 17-49
17-43 PCI Express Bus Cache Line Size Register ........ 17-49
17-44 PCI Express Bus Latency Timer Register ........ 17-50
17-45 PCI Express Bus Latency Timer Register ........ 17-50
17-46 PCI Express PCI-Compatible Configuration Header—Type 0 ........ 17-51
17-47 PCI Express Base Address Register 0 (PEXCSRBAR) ........ 17-52
17-48 32-Bit Memory Base Address Register (BAR1) ........ 17-52
17-49 64-Bit Low Memory Base Address Register ........ 17-53
17-50 64-Bit High Memory Base Address Register ........ 17-53
17-51 PCI Express Subsystem Vendor ID Register ........ 17-54
17-52 PCI Express Subsystem ID Register ........ 17-54
17-53 Capabilities Pointer Register ........ 17-55
17-54 PCI Express Interrupt Line Register ........ 17-55
17-55 PCI Express Interrupt Pin Register ........ 17-56
17-56 PCI Express Maximum Grant Register (MAX_GNT) ........ 17-56
17-57 PCI Express Maximum Latency Register (MAX_LAT) ........ 17-57
17-58 PCI Express PCI-Compatible Configuration Header—Type 1 ........ 17-57
17-59 PCI Express Base Address Register 0 (PEXCSRBAR) ........ 17-58
17-60 PCI Express Primary Bus Number Register ........ 17-58
17-61 PCI Express Secondary Bus Number Register ........ 17-59
17-62 PCI Express Subordinate Bus Number Register ........ 17-59
17-63 PCI Express I/O Base Register ........ 17-60
17-64 PCI Express I/O Limit Register ........ 17-60
17-65 PCI Express Secondary Status Register ........ 17-61
17-66 PCI Express Memory Base Register ........ 17-62

# Figures

| Figure Number | Title | Page Number |
|---|---|---|
| 17-67 | PCI Express Memory Limit Register | 17-62 |
| 17-68 | PCI Express Prefetchable Memory Base Register | 17-63 |
| 17-69 | PCI Express Prefetchable Memory Limit Register | 17-63 |
| 17-70 | PCI Express Prefetchable Base Upper 32 Bits Register | 17-64 |
| 17-71 | PCI Express Prefetchable Limit Upper 32 Bits Register | 17-64 |
| 17-72 | PCI Express I/O Base Upper 16 Bits Register | 17-64 |
| 17-73 | PCI Express I/O Limit Upper 16 Bits Register | 17-65 |
| 17-74 | Capabilities Pointer Register | 17-65 |
| 17-75 | PCI Express Interrupt Line Register | 17-66 |
| 17-76 | PCI Express Interrupt Pin Register | 17-66 |
| 17-77 | PCI Express Bridge Control Register | 17-67 |
| 17-78 | PCI Compatible Device-Specific Configuration Space | 17-68 |
| 17-79 | PCI Express Power Management Capability ID Register | 17-69 |
| 17-80 | PCI Express Power Management Capabilities Register | 17-69 |
| 17-81 | PCI Express Power Management Status and Control Register | 17-70 |
| 17-82 | PCI Express Power Management Data Register | 17-70 |
| 17-83 | PCI Express Capability ID Register | 17-71 |
| 17-84 | PCI Express Capabilities Register | 17-71 |
| 17-85 | PCI Express Device Capabilities Register | 17-72 |
| 17-86 | PCI Express Device Control Register | 17-72 |
| 17-87 | PCI Express Device Status Register | 17-73 |
| 17-88 | PCI Express Link Capabilities Register | 17-74 |
| 17-89 | PCI Express Link Control Register | 17-74 |
| 17-90 | PCI Express Link Status Register | 17-75 |
| 17-91 | PCI Express Slot Capabilities Register | 17-76 |
| 17-92 | PCI Express Slot Control Register | 17-77 |
| 17-93 | PCI Express Slot Status Register | 17-77 |
| 17-94 | PCI Express Root Control Register | 17-78 |
| 17-95 | PCI Express Root Status Register | 17-79 |
| 17-96 | PCI Express Capability ID Register | 17-79 |
| 17-97 | PCI Express MSI Message Control Register | 17-80 |
| 17-98 | PCI Express MSI Message Address Register | 17-80 |
| 17-99 | PCI Express MSI Message Upper Address Register | 17-81 |
| 17-100 | PCI Express MSI Message Data Register | 17-81 |
| 17-101 | PCI Express Extended Configuration Space | 17-82 |
| 17-102 | PCI Express Advanced Error Reporting Capability ID Register | 17-83 |
| 17-103 | PCI Express Uncorrectable Error Status Register | 17-83 |
| 17-104 | PCI Express Uncorrectable Error Mask Register | 17-84 |
| 17-105 | PCI Express Uncorrectable Error Severity Register | 17-85 |
| 17-106 | PCI Express Correctable Error Status Register | 17-86 |
| 17-107 | PCI Express Correctable Error Mask Register | 17-86 |

# Figures

Figure Number | Title | Page Number

17-108 PCI Express Advanced Error Capabilities and Control Register ........ 17-87
17-109 PCI Express Header Log Register ........ 17-88
17-110 PCI Express Root Error Command Register ........ 17-89
17-111 PCI Express Root Error Status Register ........ 17-89
17-112 PCI Express Correctable Error Source ID Register ........ 17-90
17-113 PCI Express Correctable Error Source ID Register ........ 17-90
17-114 PCI Express LTSSM State Status Register (PEX_LTSSM_STAT) ........ 17-91
17-115 PCI Express IP Block Core Clock Ratio Register (PEX_GCLK_RATIO) ........ 17-93
17-116 PCI Express Power Management Timer Register (PEX_PM_TIMER) ........ 17-93
17-117 PCI Express PME Time-Out Register (PEX_PME_TIMEOUT) ........ 17-94
17-118 PCI Express Subsystem Vendor ID Update Register (PEX_SSVID_UPDATE) ........ 17-95
17-119 PCI Express Configuration Ready Register (PEX_CFG_READY) ........ 17-95
17-120 PCI Express PME_To_Ack Timeout Register (PEX_PME_TO_ACK_TOR) ........ 17-96
17-121 PCI Express PCI Interrupt Mask Register (PEX_SS_INTR_MASK) ........ 17-97
17-122 Requestor/Completer Relationship ........ 17-97
17-123 PCI Express High-Level Layering ........ 17-98
17-124 PCI Express Packet Flow ........ 17-98
17-125 Address Invariant Byte Ordering—4 bytes Outbound ........ 17-100
17-126 Address Invariant Byte Ordering—4 bytes Inbound ........ 17-100
17-127 Address Invariant Byte Ordering—8 bytes Outbound ........ 17-100
17-128 Address Invariant Byte Ordering—2 bytes Inbound ........ 17-101
17-129 PEX_CONFIG_DATA Byte Ordering ........ 17-101
17-130 PCI Express Error Classification ........ 17-107
17-131 PCI Express Device Error Signaling Flowchart ........ 17-108
17-132 $\overline{\text{WAKE}}$ Generation Example ........ 17-116
18-1 eSPI Block Diagram ........ 18-1
18-2 Single-Master/Multi-Slave Configuration ........ 18-3
18-3 eSPI Mode Register (SPMODE) ........ 18-6
18-4 eSPI Event Register (SPIE) ........ 18-7
18-5 eSPI Mask Register (SPIM) ........ 18-8
18-6 eSPI Command Register (SPCOM) ........ 18-9
18-7 eSPI Transmit Data Register (SPITF) ........ 18-10
18-8 SPITF Example—SPMODEx[REVx]=0, SPMODEx[LENx]=3, LSB Sent First ........ 18-10
18-9 SPITF Example—SPMODEx[REVx]=x, SPMODEx[LENx]=7 ........ 18-10
18-10 SPITF Example—SPMODEx[REVx]=0, SPMODEx[LENx]=10, LSB Sent First ........ 18-10
18-11 SPITF Example—SPMODEx[REVx]=0, SPMODEx[LENx]=15, LSB Sent First ........ 18-11
18-12 SPITF Example—SPMODEx[REVx]=1, SPMODEx[LENx]=15, MSB Sent First ........ 18-11
18-13 eSPI Receive Data Register (SPIRF) ........ 18-11
18-14 SPIRF Example—SPMODEx[LENx]=3 ........ 18-11
18-15 SPIRF Example—SPMODEx[LENx]=10 ........ 18-11
18-16 SPIRF Example—SPMODEx[LENx]=15 ........ 18-11

# Figures

Figure Number | Title | Page Number

18-17 eSPI CS*n* Mode Register (SPMODE*n*) 18-12
18-18 eSPI Transfer Format with SPMODEx[CPx] = 0 18-13
18-19 eSPI Transfer Format with SPMODEx[CPx] = 1 18-14
19-1 SATA Block Diagram 19-2
19-2 Command Queue Register (CQR) 19-6
19-3 Command Active Register (CAR) 19-6
19-4 Command Completed Register (CCR) 19-7
19-5 Command Error Register (CER) 19-7
19-6 Device Error Register (DER) 19-8
19-7 Command Header Base Address Register (CHBA) 19-8
19-8 Host Status Register (HStatus) 19-9
19-9 Host Control (HControl) Register 19-12
19-10 Port Number Queue Register (CQPMP) 19-13
19-11 Signature Register (SIG) 19-14
19-12 Interrupt Coalescing Control Register (ICC) 19-14
19-13 SATA Interface Status Register (SStatus) 19-15
19-14 SATA Interface Error Register (SError) 19-16
19-15 SATA Interface Control Register (SControl) 19-18
19-16 SATA Interface Notification Register (SNotification) 19-20
19-17 Transport Layer Configuration Register (TransCfg) 19-20
19-18 Transport Layer Status Register (TransStatus) 19-21
19-19 Link Layer Configuration Register (LinkCfg) 19-21
19-20 Link Layer Configuration Register1 (LinkCfg1) 19-22
19-21 Link Layer Configuration Register1 (LinkCfg1) 19-23
19-22 Link Layer Status Register (LinkStatus) 19-23
19-23 Link Layer Status Register1 (LinkStatus1) 19-24
19-24 PHY Control Configuration Register1 (PhyCtrlCfg1) 19-26
19-25 Link Layer Command Status Register (CommandStatus) 19-27
19-26 System Priority Register (SYSPR) 19-28
19-27 Command Header 19-29
19-28 Command Descriptor 19-31
19-29 Register Host-to-Device 19-32
19-30 Register Host-to-Device First Party DMA Commands NCQ 19-32
19-31 Register Device-to-Host 19-32
19-32 Vendor-Specific BIST Operation 19-35
19-33 PRD Entry 19-43
20-1 System Connection of the eSDHC 20-1
20-2 eSDHC Block Diagram 20-2
20-3 DMA System Address Register (DSADDR) 20-6
20-4 Block Attributes Register (BLKATTR) 20-6
20-5 Command Argument Register (CMDARG) 20-7

# Figures

**Figure Number** **Title** **Page Number**

20-6 Transfer Type Register (XFERTYP) ........ 20-8
20-7 Command Response 0–3 Register (CMDRSP*n*) ........ 20-11
20-8 Buffer Data Port Register (DATPORT) ........ 20-12
20-9 Present State Register (PRSSTAT) ........ 20-13
20-10 Protocol Control Register (PROCTL) ........ 20-17
20-11 System Control Register (SYSCTL) ........ 20-19
20-12 Interrupt Status Register (IRQSTAT) ........ 20-23
20-13 Interrupt Status Enable Register (IRQSTATEN) ........ 20-27
20-14 Interrupt Signal Enable Register (IRQSIGEN) ........ 20-29
20-15 Auto CMD12 Error Status Register (AUTOC12ERR) ........ 20-31
20-16 Host Capabilities Register (HOSTCAPBLT) ........ 20-33
20-17 Watermark Level Register (WML) ........ 20-34
20-18 Force Event Register (FEVT) ........ 20-35
20-19 Host Controller Version Register (HOSTVER) ........ 20-36
20-20 DMA Control Register (DCR) ........ 20-37
20-21 eSDHC Buffer Scheme ........ 20-38
20-22 DMA CCB Interface Block ........ 20-40
20-23 Command CRC Shift Register ........ 20-42
20-24 Two Stages of Clock Divider ........ 20-43
20-25 Flow Diagram for Card Detection ........ 20-46
20-26 Flow Chart for Reset of eSDHC and SD I/O Card ........ 20-47
21-1 USB Interface Block Diagram ........ 21-1
21-2 Capability Registers Length (CAPLENGTH) ........ 21-7
21-3 Host Controller Interface Version (HCIVERSION) ........ 21-7
21-4 Host Controller Structural Parameters (HCSPARAMS) ........ 21-8
21-5 Host Control Capability Parameters (HCCPARAMS) ........ 21-8
21-6 Device Interface Version (DCIVERSION) ........ 21-9
21-7 Device Control Capability Parameters (DCCPARAMS) ........ 21-10
21-8 USB Command Register (USBCMD) ........ 21-11
21-9 USB Status Register (USBSTS) ........ 21-13
21-10 USB Interrupt Enable (USBINTR) ........ 21-16
21-11 USB Frame Index (FRINDEX) ........ 21-17
21-12 Periodic Frame List Base Address (PERIODICLISTBASE) ........ 21-19
21-13 Device Address (DEVICEADDR) ........ 21-19
21-14 Current Asynchronous List Address (ASYNCLISTADDR) ........ 21-20
21-15 Endpoint List Address (ENDPOINTLISTADDR) ........ 21-20
21-16 Master Interface Data Burst Size (BURSTSIZE) ........ 21-21
21-17 Transmit FIFO Tuning Controls (TXFILLTUNING) ........ 21-22
21-18 ULPI Register Access (ULPI VIEWPORT) ........ 21-23
21-19 Configure Flag Register (CONFIGFLAG) ........ 21-24
21-20 Port Status and Control (PORTSC) ........ 21-25

# Figures

Figure Number | Title | Page Number

21-21 USB Mode (USBMODE) .......... 21-29
21-22 Endpoint Setup Status (ENDPTSETUPSTAT) .......... 21-30
21-23 Endpoint Initialization (ENDPTPRIME) .......... 21-31
21-24 Endpoint Flush (ENDPTFLUSH) .......... 21-32
21-25 Endpoint Status (ENDPTSTATUS) .......... 21-32
21-26 Endpoint Complete (ENDPTCOMPLETE) .......... 21-33
21-27 Endpoint Control 0 (ENDPTCTRL0) .......... 21-34
21-28 Endpoint Control 1 to 5 (ENDPTCTRL*n*) .......... 21-35
21-29 Snoop 1 and Snoop 2 (SNOOP*n*) .......... 21-36
21-30 Age Count Threshold (AGE_CNT_THRESH) .......... 21-38
21-31 Priority Control (PRI_CTRL) .......... 21-38
21-32 System Interface Control Register (SI_CTRL) .......... 21-39
21-33 USB General-Purpose Register (CONTROL) .......... 21-39
21-34 Periodic Schedule Organization .......... 21-42
21-35 Frame List Link Pointer Format .......... 21-43
21-36 Asynchronous Schedule Organization .......... 21-43
21-37 Isochronous Transaction Descriptor (iTD) .......... 21-44
21-38 Split-Transaction Isochronous Transaction Descriptor (siTD) .......... 21-48
21-39 Queue Element Transfer Descriptor (qTD) .......... 21-52
21-40 Queue Head Layout .......... 21-56
21-41 Frame Span Traversal Node Structure .......... 21-60
21-42 Derivation of Pointer into Frame List Array .......... 21-66
21-43 General Format of Asynchronous Schedule List .......... 21-67
21-44 Frame Boundary Relationship Between HS Bus and FS/LS Bus .......... 21-67
21-45 Relationship of Periodic Schedule Frame Boundaries to Bus Frame Boundaries .......... 21-68
21-46 Example Periodic Schedule .......... 21-70
21-47 Example Association of iTDs to Client Request Buffer .......... 21-73
21-48 Generic Queue Head Unlink Scenario .......... 21-78
21-49 Asynchronous Schedule List with Annotation to Mark Head of List .......... 21-79
21-50 Example Mapping of qTD Buffer Pointers to Buffer Pages .......... 21-81
21-51 Host Controller Asynchronous Schedule Split-Transaction State Machine .......... 21-84
21-52 Split Transaction, Interrupt Scheduling Boundary Conditions .......... 21-87
21-53 General Structure of EHCI Periodic Schedule Utilizing Interrupt Spreading .......... 21-88
21-54 Example Host Controller Traversal of Recovery Path via FSTNs .......... 21-90
21-55 Split Transaction State Machine for Interrupt .......... 21-93
21-56 Split Transaction, Isochronous Scheduling Boundary Conditions .......... 21-99
21-57 siTD Scheduling Boundary Examples .......... 21-101
21-58 Split Transaction State Machine for Isochronous .......... 21-104
21-59 End Point Queue Head Organization .......... 21-117
21-60 Endpoint Queue Head Layout .......... 21-118
21-61 Endpoint Transfer Descriptor (dTD) .......... 21-120

# Figures

| Figure Number | Title | Page Number |
|---|---|---|
| 21-62 | USB 2.0 Device States | 21-124 |
| 21-63 | Endpoint Queue Head Diagram | 21-136 |
| 21-64 | Software Link Pointers | 21-138 |
| 21-65 | ULPI Timing | 21-148 |
| 21-66 | Sending of RX CMD | 21-149 |
| 21-67 | ULPI Data Transmit (NOPID) | 21-149 |
| 21-68 | ULPI Data Transmit (PID) | 21-149 |
| 21-69 | ULPI Data Receive | 21-150 |
| 21-70 | ULPI Register Write | 21-150 |
| 21-71 | ULPI Register Read | 21-150 |
| 22-1 | GPIO Module Block Diagram | 22-1 |
| 22-2 | GPIO Direction Register (GPDIR) | 22-2 |
| 22-3 | GPIO Open Drain Register (GPODR) | 22-3 |
| 22-4 | GPIO Data Register (GPDAT) | 22-3 |
| 22-5 | GPIO Interrupt Event Register (GPIER) | 22-4 |
| 22-6 | GPIO Interrupt Mask Register (GPIMR) | 22-4 |
| 22-7 | GPIO Interrupt Control Register (GPICR) | 22-5 |
| 23-1 | POR PLL Status Register (PORPLLSR) | 23-5 |
| 23-2 | POR Boot Mode Status Register (PORBMSR) | 23-7 |
| 23-3 | POR I/O Impedance Status and Control Register (PORIMPSCR) | 23-8 |
| 23-4 | POR Device Status Register (PORDEVSR) | 23-9 |
| 23-5 | POR Debug Mode Status Register (PORDBGMSR) | 23-12 |
| 23-6 | POR Device Status Register 2 (PORDEVSR2) | 23-13 |
| 23-7 | POR Configuration Register (GPPORCR) | 23-13 |
| 23-8 | General Configuration Register (GENCFGR) | 23-14 |
| 23-9 | Alternate Function Pin Multiplex Control Register (PMUXCR) | 23-15 |
| 23-10 | Device Disable Register (DEVDISR) | 23-17 |
| 23-11 | Power Management Jog Control Register (PMJCR) | 23-19 |
| 23-12 | Power Management Control and Status Register (POWMGTCSR) | 23-21 |
| 23-13 | Power Management Reset Counters Configuration Register (PMRCCR) | 23-22 |
| 23-14 | Power Management Power Down Counter Configuration Register (PMPDCCR) | 23-24 |
| 23-15 | Power Management Clock Disable Register (PMCDR) | 23-25 |
| 23-16 | Machine Check Summary Register (MCPSUMR) | 23-27 |
| 23-17 | Reset Request Status and Control Register (RSTRSCR) | 23-27 |
| 23-18 | Exception Reset Control Register (ECTRSTCR) | 23-28 |
| 23-19 | Automatic Reset Status Register (AUTORSTSR) | 23-28 |
| 23-20 | Processor Version Register (PVR) | 23-29 |
| 23-21 | System Version Register (SVR) | 23-30 |
| 23-22 | Reset Control Register (RSTCR) | 23-30 |
| 23-23 | LBC Voltage Select Control Register (LBCVSELCR) | 23-31 |
| 23-24 | DDR Clock Disable Register (DDRCLKDR) | 23-32 |

# Figures

**Figure Number** **Title** **Page Number**

23-25 Clock Out Control Register (CLKOCR) .......... 23-33
23-26 ECM Control Register (ECMCR) .......... 23-33
23-27 General Control Register (GCR) .......... 23-34
23-28 SerDes1 Control Register 0 (SRDS1CR0) .......... 23-35
23-29 SerDes1 Control Register 2 (SRD1SCR2) .......... 23-37
23-30 SerDes2 Control Register 0 (SRDS2CR0) .......... 23-39
23-31 SerDes2 Control Register 1 (SRDS2CR1) .......... 23-41
23-32 SerDes2 Control Register 2 (SRDSCR2) .......... 23-42
23-33 SerDes2 Control Register 3 (SRDS2CR3) .......... 23-44
23-34 e500 Core Power Management State Diagram .......... 23-48
23-35 MPC8536E Power Management Handshaking Signals .......... 23-53
23-36 Power Supply Switch for MPC8536E .......... 23-60
24-1 Performance Monitor Block Diagram .......... 24-2
24-2 Performance Monitor Global Control Register (PMGC0) .......... 24-5
24-3 Performance Monitor Local Control Register A0 (PMLCA0) .......... 24-6
24-4 Performance Monitor Local Control A Registers (PMLCA1–PMLCA9) .......... 24-7
24-5 Performance Monitor Local Control Register B0 (PMLCB0) .......... 24-8
24-6 Performance Monitor Local Control Register B (PMLCB1–PMLCB9) .......... 24-9
24-7 Performance Monitor Counter Register 0 (PMC0) .......... 24-10
24-8 Performance Monitor Counter Register (PMC1–PMC9) .......... 24-11
24-9 Duration Threshold Event Sequence Timing Diagram .......... 24-13
24-10 Burst Size, Distance, Granularity, and Burstiness Counting .......... 24-14
24-11 Burstiness Counting Timing Diagram .......... 24-16
25-1 Debug and Watchpoint Monitor Block Diagram .......... 25-2
25-2 Watchpoint Monitor Control Register 0 (WMCR0) .......... 25-10
25-3 Watchpoint Monitor Control Register 1 (WMCR1) .......... 25-12
25-4 Watchpoint Monitor Address Register (WMAR) .......... 25-12
25-5 Watchpoint Monitor Address Mask Register (WMAMR) .......... 25-13
25-6 Watchpoint Monitor Transaction Mask Register (WMTMR) .......... 25-13
25-7 Watchpoint Monitor Status Register (WMSR) .......... 25-15
25-8 Trace Buffer Control Register 0 (TBCR0) .......... 25-15
25-9 Trace Buffer Control Register 1 (TBCR1) .......... 25-17
25-10 Trace Buffer Address Register (TBAR) .......... 25-18
25-11 Trace Buffer Address Mask Register (TBAMR) .......... 25-18
25-12 Trace Buffer Transaction Mask Register (TBTMR) .......... 25-19
25-13 Trace Buffer Status Register (TBSR) .......... 25-19
25-14 Trace Buffer Access Control Register (TBACR) .......... 25-20
25-15 Trace Buffer Read High Register (TBADHR) .......... 25-21
25-16 Trace Buffer Access Data Register (TBADR) .......... 25-22
25-17 Programmed Context ID Register (PCIDR) .......... 25-22
25-18 Current Context ID Register (CCIDR) .......... 25-23

# Figures

| Figure Number | Title | Page Number |
|---|---|---|
| 25-19 | Trigger Out Source Register (TOSR) | 25-23 |
| 25-20 | e500 Coherency Module Dispatch (CMD) Trace Buffer Entry | 25-27 |
| 25-21 | DDR Trace Buffer Entry | 25-28 |
| 25-22 | PCI Trace Buffer Entry | 25-28 |
| 25-23 | PCI Express Trace Buffer Entry | 25-29 |
| C-1 | MPC8535E Block Diagram | C-2 |

# Tables

| Table Number | Title | Page Number |
|---|---|---|
| 1-1 | Supported SerDes 1 (PCI Express) Configurations | 1-5 |
| 1-2 | Supported eTSEC1 and eTSEC3 Configurations | 1-5 |
| 2-1 | Target Interface Codes | 2-1 |
| 2-2 | Local Access Windows Example | 2-2 |
| 2-3 | Format of ATMU Window Definitions | 2-3 |
| 2-4 | Local Access Register Memory Map | 2-5 |
| 2-5 | LAIPBRR1 Field Descriptions | 2-6 |
| 2-6 | LAIPBRR2 Field Descriptions | 2-6 |
| 2-7 | LAWBAR*n* Field Descriptions | 2-7 |
| 2-8 | LAWAR*n* Field Descriptions | 2-8 |
| 2-9 | Overlapping Local Access Windows | 2-8 |
| 2-10 | Local Memory Configuration, Control, and Status Register Summary | 2-11 |
| 2-11 | CCSR Block Base Address Map | 2-15 |
| 3-1 | MPC8536E Signal Reference by Functional Block | 3-2 |
| 3-2 | MPC8536E Alphabetical Signal Reference | 3-9 |
| 3-3 | MPC8536E Reset Configuration Signals | 3-16 |
| 3-4 | Output Signal States During System Reset | 3-17 |
| 4-1 | Signal Summary | 4-1 |
| 4-2 | System Control Signals—Detailed Signal Descriptions | 4-2 |
| 4-3 | Clock Signals—Detailed Signal Descriptions | 4-3 |
| 4-4 | Local Configuration Control Register Map | 4-4 |
| 4-5 | CCSRBAR Bit Settings | 4-5 |
| 4-6 | ALTCBAR Bit Settings | 4-6 |
| 4-7 | ALTCAR Bit Settings | 4-6 |
| 4-8 | BPTR Bit Settings | 4-8 |
| 4-9 | CCB Clock PLL Ratio | 4-12 |
| 4-10 | e500 Core Clock PLL Ratios | 4-12 |
| 4-11 | DDR Complex Clock PLL Ratios | 4-13 |
| 4-12 | System Speed Configuration | 4-13 |
| 4-13 | Core Speed Configuration | 4-14 |
| 4-14 | Boot ROM Location | 4-15 |
| 4-15 | Host/Agent Configuration | 4-16 |
| 4-16 | SerDes1 I/O Port Selection | 4-16 |
| 4-17 | SerDes2 I/O Port Selection | 4-17 |
| 4-18 | CPU Boot Configuration | 4-18 |
| 4-19 | Boot Sequencer Configuration | 4-18 |
| 4-20 | DDR DRAM Type | 4-19 |
| 4-21 | Serdes 2 Reference Clock Configuration | 4-19 |

# Tables

Table Number | Title | Page Number

4-22 eTSEC1 Width Configuration ... 4-20
4-23 eTSEC3 Width Configuration ... 4-20
4-24 eTSEC1 Protocol Configuration ... 4-21
4-25 eTSEC3 Protocol Configuration ... 4-21
4-26 PCI Clock Select ... 4-22
4-27 PCI Speed Configuration ... 4-22
4-28 PCI I/O Impedance ... 4-22
4-29 PCI Arbiter Configuration ... 4-23
4-30 Memory Debug Configuration ... 4-23
4-31 DDR Debug Configuration ... 4-23
4-32 General-Purpose POR Configuration ... 4-24
4-33 Engineering Use POR Configuration ... 4-24
4-34 High Speed Interface Clocking ... 4-25
4-35 SD/MMC Card Data Structure ... 4-29
4-36 Config Address Field Description, CNT = 0 ... 4-31
4-37 Config Address Field Description, CNT = 1 ... 4-32
4-38 eSPI EEPROM Data Structure ... 4-37
4-39 Config Address Field Description, CNT = 0 ... 4-39
4-40 Config Address Field Description, CNT = 1 ... 4-39
5-1 Differences Between the e500 Core and the PowerQUICC III Core Implementation ... 5-4
5-2 Device Revision Level Cross-Reference ... 5-6
6-1 Available L2 Cache/SRAM Configurations ... 6-3
6-2 Way Selection for SRAM Accesses ... 6-6
6-3 L2/SRAM Memory-Mapped Registers ... 6-9
6-4 L2CTL Field Descriptions ... 6-10
6-5 L2CEWAR*n* Field Descriptions ... 6-14
6-6 L2CEWAREA*n* Field Descriptions ... 6-14
6-7 L2CEWCR*n* Field Descriptions ... 6-15
6-8 L2SRBAR*n* Field Descriptions ... 6-16
6-9 L2SRBAREA*n* Field Descriptions ... 6-17
6-10 L2ERRINJHI Field Description ... 6-18
6-11 L2ERRINJLO Field Description ... 6-19
6-12 L2ERRINJCTL Field Descriptions ... 6-19
6-13 L2CAPTDATAHI Field Description ... 6-20
6-14 L2CAPTDATALO Field Description ... 6-20
6-15 L2CAPTECC Field Descriptions ... 6-21
6-16 L2ERRDET Field Descriptions ... 6-21
6-17 L2ERRDIS Field Descriptions ... 6-22
6-18 L2ERRINTEN Field Descriptions ... 6-23
6-19 L2ERRATTR Field Descriptions ... 6-23
6-20 L2ERRADDRL Field Description ... 6-24

# Tables

Table Number | Title | Page Number

6-21 L2ERRADDRH Field Description ... 6-25
6-22 L2ERRCTL Field Descriptions ... 6-25
6-23 Fastest Read Timing—Hit in L2 ... 6-27
6-24 PLRU Bit Update Algorithm ... 6-32
6-25 PLRU-Based Victim Selection Mechanism ... 6-33
6-26 L2 Cache States ... 6-35
6-27 State Transitions Due to Core-Initiated Transactions ... 6-35
6-28 State Transitions Due to System-Initiated Transactions ... 6-38
6-29 L2 Cache ECC Syndrome Encoding ... 6-39
6-30 L2 Cache ECC Syndrome Encoding (Check Bits) ... 6-40
7-1 ECM Memory Map ... 7-3
7-2 EEBACR Field Descriptions ... 7-4
7-3 EEBPCR Field Descriptions ... 7-4
7-4 EIPBRR1 Field Descriptions ... 7-5
7-5 EIPBRR2 Field Descriptions ... 7-6
7-6 EEDR Field Descriptions ... 7-6
7-7 EEER Field Descriptions ... 7-7
7-8 EEATR Field Descriptions ... 7-7
7-9 EELADR Field Descriptions ... 7-8
7-10 EEHADR Field Descriptions ... 7-9
8-1 DDR Memory Interface Signal Summary ... 8-4
8-2 Memory Address Signal Mappings ... 8-5
8-3 Memory Interface Signals—Detailed Signal Descriptions ... 8-6
8-4 Clock Signals—Detailed Signal Descriptions ... 8-10
8-5 DDR Memory Controller Memory Map ... 8-10
8-6 CS*n*_BNDS Field Descriptions ... 8-13
8-7 CS*n*_CONFIG Field Descriptions ... 8-14
8-8 CSn_CONFIG_2 Field Descriptions ... 8-15
8-9 TIMING_CFG_3 Field Descriptions ... 8-16
8-10 TIMING_CFG_0 Field Descriptions ... 8-18
8-11 TIMING_CFG_1 Field Descriptions ... 8-20
8-12 TIMING_CFG_2 Field Descriptions ... 8-22
8-13 DDR_SDRAM_CFG Field Descriptions ... 8-24
8-14 DDR_SDRAM_CFG_2 Field Descriptions ... 8-27
8-15 DDR_SDRAM_MODE Field Descriptions ... 8-29
8-16 DDR_SDRAM_MODE_2 Field Descriptions ... 8-30
8-17 DDR_SDRAM_MD_CNTL Field Descriptions ... 8-31
8-18 Settings of DDR_SDRAM_MD_CNTL Fields ... 8-32
8-19 DDR_SDRAM_INTERVAL Field Descriptions ... 8-33
8-20 DDR_DATA_INIT Field Descriptions ... 8-33
8-21 DDR_SDRAM_CLK_CNTL Field Descriptions ... 8-34

# Tables

| Table Number | Title | Page Number |
|---|---|---|
| 8-22 | DDR_INIT_ADDR Field Descriptions | 8-35 |
| 8-23 | DDR_INIT_EXT_ADDR Field Descriptions | 8-35 |
| 8-24 | TIMING_CFG_4 Field Descriptions | 8-36 |
| 8-25 | TIMING_CFG_5 Field Descriptions | 8-38 |
| 8-26 | DDR_ZQ_CNTL Field Descriptions | 8-39 |
| 8-27 | DDR_WRLVL_CNTL Field Descriptions | 8-41 |
| 8-28 | DDR_SR_CNTR Field Descriptions | 8-44 |
| 8-29 | DDR_Register Control Word 1 Field Descriptions | 8-45 |
| 8-30 | DDR_Register Control Word 2 Field Descriptions | 8-45 |
| 8-31 | DDRDSR_1 Field Descriptions | 8-46 |
| 8-32 | DDRDSR_2 Field Descriptions | 8-47 |
| 8-33 | DDRCDR_1 Field Descriptions | 8-49 |
| 8-34 | DDRCDR_2 Field Descriptions | 8-50 |
| 8-35 | DDR_IP_REV1 Field Descriptions | 8-51 |
| 8-36 | DDR_IP_REV2 Field Descriptions | 8-51 |
| 8-37 | DATA_ERR_INJECT_HI Field Descriptions | 8-52 |
| 8-38 | DATA_ERR_INJECT_LO Field Descriptions | 8-52 |
| 8-39 | ERR_INJECT Field Descriptions | 8-53 |
| 8-40 | CAPTURE_DATA_HI Field Descriptions | 8-53 |
| 8-41 | CAPTURE_DATA_LO Field Descriptions | 8-54 |
| 8-42 | CAPTURE_ECC Field Descriptions | 8-54 |
| 8-43 | ERR_DETECT Field Descriptions | 8-55 |
| 8-44 | ERR_DISABLE Field Descriptions | 8-56 |
| 8-45 | ERR_INT_EN Field Descriptions | 8-57 |
| 8-46 | CAPTURE_ATTRIBUTES Field Descriptions | 8-58 |
| 8-47 | CAPTURE_ADDRESS Field Descriptions | 8-59 |
| 8-48 | CAPTURE_EXT_ADDRESS Field Descriptions | 8-59 |
| 8-49 | ERR_SBE Field Descriptions | 8-60 |
| 8-50 | Byte Lane to Data Relationship | 8-64 |
| 8-51 | Supported DDR1 SDRAM Device Configurations | 8-65 |
| 8-52 | Supported DDR2 SDRAM Device Configurations | 8-65 |
| 8-53 | Supported DDR3 SDRAM Device Configurations | 8-66 |
| 8-54 | Supported DDR2 SDRAM Device Configurations—One Physical Bank | 8-66 |
| 8-55 | DDR2 Address Multiplexing for 64-Bit Data Bus with Interleaving and Partial Array Self Refresh Disabled | 8-67 |
| 8-56 | DDR2 Address Multiplexing for 32-Bit Data Bus with Interleaving and Partial Array Self Refresh Disabled | 8-67 |
| 8-57 | Example of Address Multiplexing for 64-Bit Data Bus Interleaving between Two Banks with Partial Array Self Refresh Disabled | 8-68 |
| 8-58 | Example of Address Multiplexing for 64-Bit Data Bus Interleaving between Four Banks with Partial Array Self Refresh Disabled | 8-69 |

# Tables

| Table Number | Title | Page Number |
|---|---|---|
| 8-59 | DDR2 Address Multiplexing with Partial Array Self Refresh Enabled | 8-70 |
| 8-60 | Example of Address Multiplexing for 64-bit Data Bus Interleaving Between Two Banks with Partial Array Self Refresh Enabled | 8-70 |
| 8-61 | DDR SDRAM Command Table | 8-72 |
| 8-62 | DDR SDRAM Interface Timing Intervals | 8-73 |
| 8-63 | DDR SDRAM Power-Saving Modes Refresh Configuration | 8-81 |
| 8-64 | Memory Controller–Data Beat Ordering | 8-83 |
| 8-65 | DDR SDRAM ECC Syndrome Encoding | 8-84 |
| 8-66 | DDR SDRAM ECC Syndrome Encoding (Check Bits) | 8-86 |
| 8-67 | Memory Controller Errors | 8-87 |
| 8-68 | Memory Interface Configuration Register Initialization Parameters | 8-87 |
| 8-69 | Programming Differences Between Memory Types | 8-89 |
| 9-1 | Processor Interrupts Generated Outside the Core—Types and Sources | 9-4 |
| 9-2 | Internal Interrupt Assignments | 9-6 |
| 9-3 | Interrupt Signals—Detailed Signal Descriptions | 9-8 |
| 9-4 | PIC Register Address Map | 9-9 |
| 9-5 | BRR1 Field Descriptions | 9-19 |
| 9-6 | BRR2 Field Descriptions | 9-20 |
| 9-7 | FRR Field Descriptions | 9-20 |
| 9-8 | GCR Field Descriptions | 9-21 |
| 9-9 | VIR Field Descriptions | 9-21 |
| 9-10 | PIR Field Descriptions | 9-22 |
| 9-11 | IPIVPR*n* Field Descriptions | 9-23 |
| 9-12 | SVR Field Descriptions | 9-23 |
| 9-13 | TFRR*x* Field Descriptions | 9-24 |
| 9-14 | GTCCR*xn* Field Descriptions | 9-24 |
| 9-15 | GTBCR*xn* Field Descriptions | 9-25 |
| 9-16 | GTVPR*xn* Field Descriptions | 9-26 |
| 9-17 | GTDR*xn* Field Descriptions | 9-26 |
| 9-18 | Parameters for Hourly Interrupt Timer Cascade Example | 9-27 |
| 9-19 | TCR*x* Field Descriptions | 9-28 |
| 9-20 | ERQSR Field Descriptions | 9-29 |
| 9-21 | IRQSR0 Field Descriptions | 9-30 |
| 9-22 | IRQSR1 Field Descriptions | 9-30 |
| 9-23 | IRQSR2 Field Descriptions | 9-31 |
| 9-24 | CISR0 Field Descriptions | 9-31 |
| 9-25 | CISR1 Field Descriptions | 9-32 |
| 9-26 | CISR2 Field Descriptions | 9-32 |
| 9-27 | PM*n*MR0 Field Descriptions | 9-33 |
| 9-28 | PM*n*MR1 Field Descriptions | 9-34 |
| 9-29 | PM*n*MR2 Field Descriptions | 9-34 |

# Tables

**Table Number** | **Title** | **Page Number**

9-30 MSGR*n* Field Descriptions ........ 9-35
9-31 MER Field Descriptions ........ 9-36
9-32 MSR Field Descriptions ........ 9-36
9-33 MSIR*n* Field Descriptions ........ 9-37
9-34 MSISR Field Descriptions ........ 9-37
9-35 MSIIR Field Descriptions ........ 9-38
9-36 MSIVPRn Field Descriptions ........ 9-39
9-37 MSIDR*n* Field Descriptions ........ 9-39
9-38 EIVPR*n* Field Descriptions ........ 9-41
9-39 EIDR*n* Field Descriptions ........ 9-42
9-40 IIVPR*n* Field Descriptions ........ 9-44
9-41 IIDR*n* Field Descriptions ........ 9-45
9-42 MIVPR*n* Field Descriptions ........ 9-45
9-43 MIDR*n* Field Descriptions ........ 9-46
9-44 Per-CPU Registers—Private Access Address Offsets ........ 9-47
9-45 IPIDR*n* Field Descriptions ........ 9-49
9-46 CTPRn Field Descriptions ........ 9-49
9-47 WHOAMIn Field Descriptions ........ 9-50
9-48 IACKn Field Descriptions ........ 9-51
9-49 EOIn Field Descriptions ........ 9-51
9-50 PCI Express INTx/IRQ*n* Sharing ........ 9-56
10-1 Example Descriptor ........ 10-5
10-2 SEC Address Map ........ 10-12
10-3 SEC Address Map ........ 10-13
10-4 Header Dword Bit Definitions ........ 10-21
10-5 Header Dword Writeback Bit Definitions ........ 10-22
10-6 EU_SEL0 and EU_SEL1 Values ........ 10-23
10-7 Descriptor Types ........ 10-23
10-8 Pointer Dword Field Definitions ........ 10-25
10-9 Link Table Field Definitions ........ 10-27
10-10 Descriptor Format Summary ........ 10-30
10-11 Channel Configuration Register Fields ........ 10-38
10-12 Writeback Options ........ 10-40
10-14 Channel Status Register Field Descriptions ........ 10-41
10-13 Done Interrupt Options ........ 10-41
10-15 Channel Status Register Error Field Definitions ........ 10-43
10-16 Current Descriptor Pointer Register Fields ........ 10-44
10-17 Fetch FIFO Enqueue Register Field Descriptions ........ 10-44
10-18 Channel Assignment Value ........ 10-50
10-19 Field Names in Interrupt Enable, Interrupt Status, and Interrupt Clear Registers ........ 10-51
10-20 IP Block Revision Register Fields ........ 10-55

# Tables

| Table Number | Title | Page Number |
|---|---|---|
| 10-21 | Master Control Register Fields | 10-55 |
| 10-22 | AESU Mode Register Field Descriptions | 10-58 |
| 10-23 | AES Cipher Modes | 10-60 |
| 10-24 | Use of Data Size Register | 10-62 |
| 10-25 | AESU Reset Control Register Field Descriptions | 10-62 |
| 10-26 | AESU Status Register Field Descriptions | 10-63 |
| 10-27 | AESU Interrupt Status Register Field Descriptions | 10-64 |
| 10-28 | AESU Interrupt Mask Register Field Descriptions | 10-66 |
| 10-29 | AESU Context Registers for Confidentiality Modes | 10-69 |
| 10-30 | AESU Context Registers for Integrity Modes | 10-71 |
| 10-31 | AESU Context Registers for Modes Providing Confidentiality and Integrity | 10-74 |
| 10-32 | GCM Cipher Mode Auxiliary Bit Definitions | 10-80 |
| 10-33 | GCM Encryption Context | 10-82 |
| 10-34 | GCM Decryption Context | 10-83 |
| 10-35 | GCM with ICV Context | 10-84 |
| 10-36 | GCM-GHASH Context | 10-85 |
| 10-37 | AFEU Mode Register Field Descriptions | 10-89 |
| 10-38 | AFEU Reset Control Register Field Descriptions | 10-91 |
| 10-39 | AFEU Status Register Field Descriptions | 10-92 |
| 10-40 | AFEU Interrupt Status Register Fields | 10-93 |
| 10-41 | AFEU Interrupt Mask Register | 10-95 |
| 10-42 | CRCU Reset Control Register Field Descriptions | 10-100 |
| 10-43 | CRCU Status Register Bit Definitions | 10-102 |
| 10-44 | CRCU Interrupt Status Register Bit Definitions | 10-103 |
| 10-45 | CRCU Interrupt Mask Register Bit Definitions | 10-105 |
| 10-46 | DEU Mode Register Field Descriptions | 10-109 |
| 10-47 | DEU Cipher Modes | 10-109 |
| 10-48 | DEU Key Size Register Field Descriptions | 10-110 |
| 10-49 | DEU Reset Control Register Field Descriptions | 10-111 |
| 10-50 | DEU Status Register | 10-112 |
| 10-51 | DEU Interrupt Status Register Field Descriptions | 10-113 |
| 10-52 | DEU Interrupt Mask Register Field Descriptions | 10-115 |
| 10-53 | KEU Mode Register Field Descriptions | 10-118 |
| 10-54 | KEU Reset Control Register Field Descriptions | 10-121 |
| 10-55 | KEU Status Register Fields Description | 10-122 |
| 10-56 | KEU Interrupt Status Register Signals Description | 10-124 |
| 10-57 | KEU Interrupt Mask Register Fields Description | 10-126 |
| 10-58 | KEU IV_1 Register Fields Description | 10-128 |
| 10-59 | MDEU Mode Register in Old Configuration | 10-133 |
| 10-60 | MDEU Mode Register in New Configuration | 10-134 |
| 10-61 | Mode Register—HMAC or SSL-MAC Generated by Single Descriptor | 10-136 |

# Tables

Table Number | Title | Page Number

10-62 Mode Register—HMAC Generated Across a Sequence of Descriptors.......... 10-136
10-63 MDEU Reset Control Register Field Descriptions.......... 10-138
10-64 MDEU Status Register Field Descriptions.......... 10-139
10-65 MDEU Interrupt Status Register Field Descriptions.......... 10-140
10-66 MDEU Interrupt Mask Register Field Descriptions.......... 10-141
10-67 ROUTINE Field Description.......... 10-147
10-68 PKEU Reset Control Register Field Descriptions.......... 10-149
10-69 PKEU Status Register Field Descriptions.......... 10-150
10-70 PKEU Interrupt Status Register Field Descriptions.......... 10-152
10-71 PKEU Interrupt Mask Register Field Descriptions.......... 10-153
10-72 RNGU Reset Control Register Field Descriptions.......... 10-157
10-73 RNGU Status Register Field Descriptions.......... 10-157
10-74 RNGU Interrupt Status Register Field Descriptions.......... 10-159
10-75 RNGU Interrupt Mask Register Field Descriptions.......... 10-160
11-1 I$^2$C Interface Signal Descriptions.......... 11-3
11-2 I$^2$C Interface Signal—Detailed Signal Descriptions.......... 11-4
11-3 I$^2$C Memory Map.......... 11-5
11-4 I2CADR Field Descriptions.......... 11-6
11-5 I2CFDR Field Descriptions.......... 11-7
11-6 I2CCR Field Descriptions.......... 11-8
11-7 I2CSR Field Descriptions.......... 11-9
11-8 I2CDR Field Descriptions.......... 11-10
11-9 I2CDFSRR Field Descriptions.......... 11-11
12-1 DUART Signals—Detailed Signal Descriptions.......... 12-3
12-2 DUART Register Summary.......... 12-4
12-3 URBR Field Descriptions.......... 12-5
12-4 UTHR Field Descriptions.......... 12-6
12-5 UDMB Field Descriptions.......... 12-6
12-6 UDLB Field Descriptions.......... 12-7
12-7 Baud Rate Examples.......... 12-7
12-8 UIER Field Descriptions.......... 12-8
12-9 UIIR Field Descriptions.......... 12-9
12-10 UIIR IID Bits Summary.......... 12-9
12-11 UFCR Field Descriptions.......... 12-10
12-12 UAFR Field Descriptions.......... 12-11
12-13 ULCR Field Descriptions.......... 12-12
12-14 Parity Selection Using ULCR[PEN], ULCR[SP], and ULCR[EPS].......... 12-14
12-15 UMCR Field Descriptions.......... 12-14
12-16 ULSR Field Descriptions.......... 12-15
12-17 UMSR Field Descriptions.......... 12-16
12-18 USCR Field Descriptions.......... 12-17

# Tables

Table Number | Title | Page Number

12-19 UDSR Field Descriptions .... 12-17
12-20 UDSR[TXRDY] Set Conditions .... 12-18
12-21 UDSR[TXRDY] Cleared Conditions .... 12-18
12-22 UDSR[RXRDY] Set Conditions .... 12-18
12-23 UDSR[RXRDY] Cleared Conditions .... 12-19
13-1 Signal Properties—Summary .... 13-4
13-2 Enhanced Local Bus Controller Detailed Signal Descriptions .... 13-5
13-3 Enhanced Local Bus Controller Registers .... 13-9
13-4 BR*n* Field Descriptions .... 13-11
13-5 Reset value of OR0 Register .... 13-13
13-6 Memory Bank Sizes in Relation to Address Mask .... 13-13
13-7 OR*n*—GPCM Field Descriptions .... 13-14
13-8 OR*n*—FCM Field Descriptions .... 13-17
13-9 OR*n*—UPM Field Descriptions .... 13-20
13-10 MAR Field Descriptions .... 13-21
13-11 M*x*MR Field Descriptions .... 13-21
13-12 MRTPR Field Descriptions .... 13-23
13-13 MDR Field Description .... 13-24
13-14 LSOR Field Description .... 13-25
13-15 LURT Field Descriptions .... 13-26
13-16 LTESR Field Descriptions .... 13-27
13-17 LTEDR Field Descriptions .... 13-28
13-18 LTEIR Field Descriptions .... 13-29
13-19 LTEATR Field Descriptions .... 13-31
13-20 LTEAR Field Descriptions .... 13-31
13-21 LTECCR Field Descriptions .... 13-32
13-22 LBCR Field Descriptions .... 13-33
13-23 LCRR Field Descriptions .... 13-35
13-24 FMR Field Descriptions .... 13-36
13-25 FIR Field Descriptions .... 13-38
13-26 FCR Field Descriptions .... 13-38
13-27 FBAR Field Descriptions .... 13-39
13-28 FPAR Field Descriptions, Small Page Device (ORx[PGS] = 0) .... 13-40
13-29 FPAR Field Descriptions, Large Page Device (ORx[PGS] = 1) .... 13-40
13-30 FBCR Field Descriptions .... 13-41
13-31 FECCn Field Descriptions .... 13-42
13-32 GPCM Read Control Signal Timing .... 13-50
13-33 GPCM Write Control Signal Timing .... 13-52
13-34 Boot Bank Field Values after Reset for GPCM as Boot Controller .... 13-59
13-35 FCM Chip-Select to First Command Timing .... 13-68
13-36 FCM Command, Address, and Write Data Timing Parameters .... 13-69

# Tables

**Table Number** **Title** **Page Number**

13-37 FCM Read Data Timing Parameters ........ 13-71
13-38 Boot Bank Field Values after Reset for FCM as Boot Controller ........ 13-72
13-39 UPM Routines Start Addresses ........ 13-76
13-40 RAM Word Field Descriptions ........ 13-81
13-41 M*x*MR Loop Field Use ........ 13-86
13-42 UPM Address Multiplexing ........ 13-87
13-43 Data Bus Drive Requirements For Read Cycles ........ 13-94
13-44 FCM Register Settings for Soft Reset (ORn[PGS] = 1) ........ 13-95
13-45 FCM Register Settings for Status Read (ORn[PGS] = 1) ........ 13-96
13-46 FCM Register Settings for ID Read (ORn[PGS] = 1) ........ 13-96
13-47 FCM Register Settings for Page Read (ORn[PGS] = 1) ........ 13-97
13-48 FCM Register Settings for Block Erase (ORn[PGS] = 1) ........ 13-98
13-49 FCM Register Settings for Page Program (ORn[PGS] = 1) ........ 13-99
14-1 eTSEC*n* Network Interface Signal Properties ........ 14-7
14-2 eTSEC Signals—Detailed Signal Descriptions ........ 14-10
14-3 Module Memory Map Summary ........ 14-15
14-4 Module Memory Map ........ 14-16
14-5 TSEC_ID Field Descriptions ........ 14-26
14-6 TSEC_ID2 Field Descriptions ........ 14-27
14-7 TSEC_ID2[TSEC_INT] Field Settings ........ 14-27
14-8 IEVENT Field Descriptions ........ 14-29
14-9 IMASK Field Descriptions ........ 14-32
14-10 EDIS Field Descriptions ........ 14-34
14-11 ECNTRL Field Descriptions ........ 14-35
14-12 eTSEC Interface Configurations ........ 14-37
14-13 PTV Field Descriptions ........ 14-38
14-14 DMACTRL Field Descriptions ........ 14-38
14-15 TBIPA Field Descriptions ........ 14-40
14-16 TCTRL Field Descriptions ........ 14-41
14-17 TSTAT Field Descriptions ........ 14-43
14-18 DFVLAN Field Descriptions ........ 14-46
14-19 TXIC Field Descriptions ........ 14-47
14-20 TQUEUE Field Descriptions ........ 14-48
14-21 TR03WT Field Descriptions ........ 14-49
14-22 TR47WT Field Descriptions ........ 14-50
14-23 TBDBPH Field Descriptions ........ 14-50
14-24 TBPTR*n* Field Descriptions ........ 14-51
14-25 TBASEH Field Descriptions ........ 14-51
14-26 TBASE0–TBASE7 Field Descriptions ........ 14-52
14-27 TMR_TXTS*n*_ID Register Field Descriptions ........ 14-53
14-28 TMR_TXTS*n*_H/L Register Field Descriptions ........ 14-53

# Tables

| Table Number | Title | Page Number |
|---|---|---|
| 14-29 | RCTRL Field Descriptions | 14-54 |
| 14-30 | RSTAT Field Descriptions | 14-57 |
| 14-31 | RXIC Field Descriptions | 14-59 |
| 14-32 | RQUEUE Field Descriptions | 14-60 |
| 14-33 | RBIFX Field Descriptions | 14-62 |
| 14-34 | RQFAR Field Descriptions | 14-63 |
| 14-35 | RQFCR Field Descriptions | 14-64 |
| 14-36 | RQFPR Field Descriptions | 14-66 |
| 14-37 | MRBLR Field Descriptions | 14-68 |
| 14-38 | RBDBPH Field Descriptions | 14-69 |
| 14-39 | RBPTR*n* Field Descriptions | 14-69 |
| 14-40 | RBASEH Field Descriptions | 14-70 |
| 14-41 | RBASE0–RBASE7 Field Descriptions | 14-70 |
| 14-42 | TMR_RXTS_H/L Register Field Descriptions | 14-71 |
| 14-43 | MACCFG1 Field Descriptions | 14-74 |
| 14-44 | MACCFG2 Field Descriptions | 14-76 |
| 14-45 | IPGIFG Field Descriptions | 14-78 |
| 14-46 | HAFDUP Field Descriptions | 14-79 |
| 14-47 | MAXFRM Field Descriptions | 14-80 |
| 14-48 | MIIMCFG Field Descriptions | 14-81 |
| 14-49 | MIIMCOM Descriptions | 14-81 |
| 14-50 | MIIMADD Field Descriptions | 14-82 |
| 14-51 | MIIMCON Field Descriptions | 14-83 |
| 14-52 | MIIMSTAT Field Descriptions | 14-83 |
| 14-53 | MIIMIND Field Descriptions | 14-84 |
| 14-54 | IFSTAT Field Descriptions | 14-84 |
| 14-55 | MACSTNADDR1 Field Descriptions | 14-85 |
| 14-56 | MACSTNADDR2 Field Descriptions | 14-85 |
| 14-57 | MAC*n*ADDR1 Field Descriptions | 14-86 |
| 14-58 | MAC01ADDR2–MAC15ADDR2 Field Descriptions | 14-87 |
| 14-59 | TR64 Field Descriptions | 14-88 |
| 14-60 | TR127 Field Descriptions | 14-88 |
| 14-61 | TR255 Field Descriptions | 14-89 |
| 14-62 | TR511 Field Descriptions | 14-89 |
| 14-63 | TR1K Field Descriptions | 14-90 |
| 14-64 | TRMAX Field Descriptions | 14-90 |
| 14-65 | TRMGV Field Descriptions | 14-91 |
| 14-66 | RBYT Field Descriptions | 14-91 |
| 14-67 | RPKT Field Descriptions | 14-92 |
| 14-68 | RFCS Field Descriptions | 14-92 |
| 14-69 | RMCA Field Descriptions | 14-93 |

# Tables

**Table Number** | **Title** | **Page Number**

14-70 RBCA Field Descriptions 14-93
14-71 RXCF Field Descriptions 14-94
14-72 RXPF Field Descriptions 14-94
14-73 RXUO Field Descriptions 14-95
14-74 RALN Field Descriptions 14-95
14-75 RFLR Field Descriptions 14-96
14-76 RCDE Field Descriptions 14-96
14-77 RCSE Field Descriptions 14-97
14-78 RUND Field Descriptions 14-97
14-79 ROVR Field Descriptions 14-98
14-80 RFRG Field Descriptions 14-98
14-81 RJBR Field Descriptions 14-99
14-82 RDRP Field Descriptions 14-99
14-83 TBYT Field Descriptions 14-100
14-84 TPKT Field Descriptions 14-100
14-85 TMCA Field Descriptions 14-101
14-86 TBCA Field Descriptions 14-101
14-87 TXPF Field Descriptions 14-102
14-88 TDFR Field Descriptions 14-102
14-89 TEDF Field Descriptions 14-103
14-90 TSCL Field Descriptions 14-103
14-91 TMCL Field Descriptions 14-104
14-92 TLCL Field Descriptions 14-104
14-93 TXCL Field Descriptions 14-105
14-94 TNCL Field Descriptions 14-105
14-95 TDRP Field Descriptions 14-106
14-96 TJBR Field Descriptions 14-106
14-97 TFCS Field Descriptions 14-107
14-98 TXCF Field Descriptions 14-107
14-99 TOVR Field Descriptions 14-108
14-100 TUND Field Descriptions 14-108
14-101 TFRG Field Descriptions 14-109
14-102 CAR1 Field Descriptions 14-109
14-103 CAR2 Field Descriptions 14-111
14-104 CAM1 Field Descriptions 14-112
14-105 CAM2 Field Descriptions 14-113
14-106 RREJ Field Descriptions 14-115
14-107 IGADDR*n* Field Descriptions 14-116
14-108 GADDR*n* Field Descriptions 14-116
14-109 FIFOCFG Field Descriptions 14-117
14-110 ATTR Field Descriptions 14-119

# Tables

| Table Number | Title | Page Number |
|---|---|---|
| 14-111 | ATTRELI Field Descriptions | 14-120 |
| 14-112 | RQPRM Field Descriptions | 14-121 |
| 14-113 | RFBPTR0–RFBPTR7 Field Descriptions | 14-121 |
| 14-114 | TMR_CTRL Register Field Descriptions | 14-122 |
| 14-115 | TMR_TEVENT Register Field Descriptions | 14-125 |
| 14-116 | TMR_TEMASK Register Definition | 14-126 |
| 14-117 | TMR_TEMASK Register Field Descriptions | 14-126 |
| 14-118 | TMR_PEVENT Register Field Descriptions | 14-127 |
| 14-119 | TMR_PEMASK Register Field Descriptions | 14-128 |
| 14-120 | TMR_STAT Register Definition | 14-128 |
| 14-121 | TMR_STAT Register Field Descriptions | 14-128 |
| 14-122 | TMR_CNT_H/L Register Field Descriptions | 14-129 |
| 14-123 | TMR_ADD Register Field Descriptions | 14-130 |
| 14-124 | TMR_ACC Register Field Descriptions | 14-130 |
| 14-125 | TMR_PRSC Register Field Descriptions | 14-131 |
| 14-126 | TMROFF_H/L Register Field Descriptions | 14-131 |
| 14-127 | TMR_ALARM*n*_H/L Register Field Descriptions | 14-132 |
| 14-128 | TMR_FIPER Register Field Descriptions | 14-133 |
| 14-129 | TMR_ETTS1-2_H Register Field Descriptions | 14-133 |
| 14-130 | TBI MII Register Set | 14-135 |
| 14-131 | CR Field Descriptions | 14-136 |
| 14-132 | SR Descriptions | 14-137 |
| 14-133 | ANA Field Descriptions | 14-138 |
| 14-134 | PAUSE Priority Resolution | 14-139 |
| 14-135 | ANLPBPA Field Descriptions | 14-140 |
| 14-136 | ANEX Field Descriptions | 14-141 |
| 14-137 | ANNPT Field Descriptions | 14-142 |
| 14-138 | ANLPANP Field Descriptions | 14-142 |
| 14-139 | EXST Field Descriptions | 14-143 |
| 14-140 | JD Field Descriptions | 14-144 |
| 14-141 | TBICON Field Descriptions | 14-145 |
| 14-142 | GMII, MII, and RMII Signals Multiplexing | 14-152 |
| 14-143 | RGMII, TBI, and RTBI Signals Multiplexing | 14-153 |
| 14-144 | RGMII and RTBI Signals Multiplexing | 14-154 |
| 14-145 | RGMII Signals Multiplexing | 14-155 |
| 14-146 | Shared Signals | 14-156 |
| 14-147 | Signal Encoding for GMII-Style 8-Bit FIFO | 14-159 |
| 14-148 | Signal Encoding for Encoded 8-Bit FIFO | 14-159 |
| 14-149 | Steps for Minimum Register Initialization | 14-160 |
| 14-150 | Custom Preamble Field Descriptions | 14-165 |
| 14-151 | Received Preamble Field Descriptions | 14-166 |

# Tables

Table Number | Title | Page Number

14-152 Flow Control Frame Structure ... 14-170
14-153 Non-Error Transmit Interrupts ... 14-172
14-154 Non-Error Receive Interrupts ... 14-172
14-155 Interrupt Coalescing Timing Threshold Ranges ... 14-173
14-156 Transmission Errors ... 14-175
14-157 Reception Errors ... 14-175
14-158 Tx Frame Control Block Description ... 14-178
14-159 Rx Frame Control Block Descriptions ... 14-180
14-160 Supported Stack L2 Ethernet Headers ... 14-182
14-161 Special Filer Rules ... 14-186
14-162 Receive Queue Filer Interrupt Events ... 14-186
14-163 Filer Table Example—802.1p Priority Filing ... 14-187
14-164 Filer Table Example—IP Diff-Serv Code Points Filing ... 14-188
14-165 Filer Table Example—TCP and UDP Port Filing ... 14-189
14-166 PTP Payload Special Fields ... 14-196
14-167 Time-Stamp Insertion Programming Requirements ... 14-197
14-168 Tx Frame Control Block Description ... 14-199
14-169 Transmit Data Buffer Descriptor (TxBD) Field Descriptions ... 14-203
14-170 Receive Buffer Descriptor Field Descriptions ... 14-206
14-171 MII Interface Mode Signal Configuration ... 14-208
14-172 Shared MII Signals ... 14-209
14-173 MII Mode Register Initialization Steps ... 14-209
14-174 GMII Interface Mode Signal Configuration ... 14-212
14-175 Shared GMII Signals ... 14-213
14-176 GMII Mode Register Initialization Steps ... 14-213
14-177 TBI Interface Mode Signal Configuration ... 14-216
14-178 Shared TBI Signals ... 14-217
14-179 TBI Mode Register Initialization Steps ... 14-217
14-180 RGMII Interface Mode Signal Configuration ... 14-220
14-181 Shared RGMII Signals ... 14-221
14-182 RGMII Mode Register Initialization Steps ... 14-221
14-183 RMII Interface Mode Signal Configuration ... 14-224
14-184 Shared RMII Signals ... 14-225
14-185 RMII Mode Register Initialization Steps ... 14-225
14-186 RTBI Interface Mode Signal Configuration ... 14-228
14-187 Shared RTBI Signals ... 14-229
14-188 RTBI Mode Register Initialization Steps ... 14-229
14-189 8-Bit FIFO Interface Mode Signal Configurations ... 14-232
14-190 8-Bit FIFO Mode Register Initialization Steps ... 14-233
14-191 SGMII Interface Signal Configuration (4-Wire) ... 14-234
14-192 SGMII Mode Register Initialization Steps ... 14-234

# Tables

**Table Number** **Title** **Page Number**

15-1 Relationship of Modes and Features ........ 15-3
15-2 DMA Mode Bit Settings ........ 15-3
15-3 DMA Signals—Detailed Signal Descriptions ........ 15-5
15-4 DMA Register Summary ........ 15-6
15-5 MR*n* Field Descriptions ........ 15-9
15-6 SR*n* Field Descriptions ........ 15-12
15-7 ECLNDARn Field Descriptions ........ 15-14
15-8 CLNDAR*n* Field Descriptions ........ 15-14
15-9 SATR*n* Field Descriptions ........ 15-15
15-10 SAR*n* Field Descriptions ........ 15-16
15-11 DATR*n* Field Descriptions ........ 15-16
15-12 DAR*n* Field Descriptions ........ 15-17
15-13 BCR*n* Field Descriptions ........ 15-18
15-14 NLNDAR*n* Field Descriptions ........ 15-18
15-15 ENLNDAR*n* Field Descriptions ........ 15-19
15-16 ECLSDAR*n* Field Descriptions ........ 15-20
15-17 CLSDAR*n* Field Descriptions ........ 15-20
15-18 ENLSDARn Field Descriptions ........ 15-21
15-19 NLSDAR*n* Field Descriptions ........ 15-21
15-20 SSR*n* Field Descriptions ........ 15-22
15-21 DSR*n* Field Descriptions ........ 15-23
15-22 DGSR Field Descriptions ........ 15-23
15-23 Channel State Table ........ 15-31
15-24 List DMA Descriptor Summary ........ 15-33
15-25 Link DMA Descriptor Summary ........ 15-33
16-1 POR Parameters for PCI Controller ........ 16-5
16-2 PCI Interface Signals—Detailed Signal Descriptions ........ 16-6
16-3 PCI Memory-Mapped Register Map ........ 16-11
16-4 PCI CFG_ADDR Field Descriptions ........ 16-14
16-5 PCI CFG_DATA Field Descriptions ........ 16-15
16-6 PCI INT_ACK Field Descriptions ........ 16-15
16-7 POTAR*n* Field Descriptions ........ 16-16
16-8 POTEAR*n* Field Descriptions ........ 16-17
16-9 POWBAR*n* Field Descriptions ........ 16-17
16-10 POWARn Field Descriptions ........ 16-18
16-11 PITAR*n* Field Descriptions ........ 16-20
16-12 PIWBAR Field Descriptions ........ 16-21
16-13 PIWBEAR Field Descriptions ........ 16-21
16-14 PIWAR*n* Field Descriptions ........ 16-22
16-15 ERR_DR Field Descriptions ........ 16-24
16-16 ERR_CAP_DR Field Descriptions ........ 16-25

# Tables

| Table Number | Title | Page Number |
|---|---|---|
| 16-17 | ERR_EN Field Descriptions | 16-27 |
| 16-18 | ERR_ATTRIB Field Descriptions | 16-27 |
| 16-19 | ERR_ADDR Field Descriptions | 16-28 |
| 16-20 | ERR_EXT_ADDR Field Descriptions | 16-28 |
| 16-21 | ERR_DL Field Description | 16-29 |
| 16-22 | ERR_DH Field Description | 16-29 |
| 16-23 | GAS_TIMR Field Descriptions | 16-29 |
| 16-24 | PCI Vendor ID Register Field Description | 16-31 |
| 16-25 | PCI Device ID Register Field Description | 16-31 |
| 16-26 | PCI Bus Command Register Field Descriptions | 16-31 |
| 16-27 | PCI Bus Status Register Field Descriptions | 16-33 |
| 16-28 | PCI Revision ID Register Field Descriptions | 16-34 |
| 16-29 | PCI Bus Programming Interface Register Field Description | 16-34 |
| 16-30 | PCI Subclass Code Register Field Description | 16-35 |
| 16-31 | PCI Bus Base Class Code Register Field Description | 16-35 |
| 16-32 | PCI Bus Cache Line Size Register Field Descriptions | 16-36 |
| 16-33 | PCI Bus Latency Timer Register Field Descriptions | 16-36 |
| 16-34 | PCSRBAR Field Descriptions | 16-37 |
| 16-35 | 32-Bit Memory Base Address Register Field Descriptions | 16-37 |
| 16-36 | 64-Bit Low Memory Base Address Register Field Descriptions | 16-38 |
| 16-37 | Bit Setting for 64-Bit High Memory Base Address Register | 16-38 |
| 16-38 | PCI Subsystem Vendor ID Register Field Description | 16-38 |
| 16-39 | PCI Subsystem ID Register Field Description | 16-39 |
| 16-40 | PCI Bus Capabilities Pointer Register Field Description | 16-39 |
| 16-41 | PCI Bus Interrupt Line Register Field Description | 16-40 |
| 16-42 | PCI Bus Interrupt Pin Register Field Description | 16-40 |
| 16-43 | PCI Bus Minimum Grant Register Field Description | 16-40 |
| 16-44 | PCI Bus Maximum Latency Register Field Description | 16-41 |
| 16-45 | PCI Bus Function Register Field Descriptions | 16-41 |
| 16-46 | PCI Bus Arbiter Configuration Register Field Descriptions | 16-42 |
| 16-47 | PCI Bus Commands | 16-46 |
| 16-48 | Supported Combinations of PCI_AD[1:0] | 16-47 |
| 16-49 | PCI Configuration Space Header Summary | 16-59 |
| 16-50 | PCI Type 0 Configuration—Device Number to AD*n* Translation | 16-62 |
| 16-51 | Special-Cycle Message Encodings | 16-64 |
| 16-52 | PCI Mode Error Actions | 16-66 |
| 16-53 | Affected Configuration Register Bits for POR | 16-67 |
| 16-54 | Power-On Reset Values for Affected Configuration Bits | 16-68 |
| 17-1 | POR Parameters for PCI Express Controller | 17-4 |
| 17-2 | PCI Express Interface Signals—Detailed Signal Descriptions | 17-5 |
| 17-3 | PCI Express Memory-Mapped Register Map | 17-6 |

# Tables

Table Number | Title | Page Number

17-4 PEX_CONFIG_ADDR Field Descriptions ........ 17-10
17-5 PEX_CONFIG_DATA Field Descriptions ........ 17-11
17-6 PEX_OTB_CPL_TOR Field Descriptions ........ 17-11
17-7 PEX_CONF_RTY_TOR Field Descriptions ........ 17-12
17-8 PEX_CONFIG Field Descriptions ........ 17-13
17-9 PEX_PME_MES_DR Field Descriptions ........ 17-13
17-10 PEX_PME_MES_DISR Field Descriptions ........ 17-15
17-11 PEX_PME_MES_IER Field Descriptions ........ 17-17
17-12 PEX_PMCR Field Descriptions ........ 17-18
17-13 PCI Express IP Block Revision Register 1 Field Descriptions ........ 17-19
17-14 PCI Express IP Block Revision Register 2 Field Descriptions ........ 17-19
17-15 PEXOTAR*n* Field Descriptions ........ 17-21
17-16 PCI Express Outbound Extended Address Translation Register *n* Field Descriptions ........ 17-21
17-17 PCI Express Outbound Window Base Address Register *n* Field Descriptions ........ 17-22
17-18 PEXOWAR*n* Field Descriptions ........ 17-23
17-19 PCI Express Inbound Translation Address Registers Field Descriptions ........ 17-26
17-20 PCI Express Inbound Window Base Address Register Field Descriptions ........ 17-27
17-21 PCI Express Inbound Window Base Extended Address Register Field Descriptions ........ 17-28
17-22 PCI Express Inbound Window Attributes Registers Field Descriptions ........ 17-28
17-23 PCI Express Error Detect Register Field Descriptions ........ 17-31
17-24 PCI Express Error Interrupt Enable Register Field Descriptions ........ 17-33
17-25 PCI Express Error Disable Register Field Descriptions ........ 17-34
17-26 PCI Express Error Capture Status Register Field Descriptions ........ 17-36
17-27 PCI Express Error Capture Register 0 Field Descriptions
Internal Source, Outbound Transaction ........ 17-37
17-28 PCI Express Error Capture Register 0 Field Descriptions
External Source, Inbound Transaction ........ 17-38
17-29 PCI Express Error Capture Register 1 Field Descriptions
Internal Source, Outbound Transaction ........ 17-39
17-30 PCI Express Error Capture Register 1 Field Descriptions
External Source, Inbound CompletionTransaction ........ 17-39
17-31 PCI Express Error Capture Register 1 Field Descriptions
External Source, Inbound Memory Request Transaction ........ 17-40
17-32 PCI Express Error Capture Register 2 Field Descriptions
Internal Source, Outbound Transaction ........ 17-40
17-33 PCI Express Error Capture Register 2 Field Descriptions
External Source, Inbound Completion Transaction ........ 17-41
17-34 PCI Express Error Capture Register 2 Field Descriptions
External Source, Inbound Memory Request Transaction ........ 17-41
17-35 PCI Express Error Capture Register 3 Field Descriptions
Internal Source, Outbound Transaction ........ 17-42

# Tables

| Table Number | Title | Page Number |
|---|---|---|
| 17-36 | PEX Error Capture Register 3 Field Descriptions External Source, Inbound Memory Request Transaction | 17-43 |
| 17-37 | PCI Express Vendor ID Register Field Description | 17-45 |
| 17-38 | PCI Express Device ID Register Field Description | 17-46 |
| 17-39 | PCI Express Command Register Field Descriptions | 17-46 |
| 17-40 | PCI Express Status Register Field Descriptions | 17-48 |
| 17-41 | PCI Express Revision ID Register Field Descriptions | 17-48 |
| 17-42 | PCI Express Class Code Register Field Descriptions | 17-49 |
| 17-43 | PCI Express Bus Cache Line Size Register Field Descriptions | 17-49 |
| 17-44 | PCI Express Bus Latency Timer Register Field Descriptions | 17-50 |
| 17-45 | PCI Express Bus Latency Timer Register Field Descriptions | 17-50 |
| 17-46 | PEXCSRBAR Field Descriptions | 17-52 |
| 17-47 | 32-Bit Memory Base Address Register (BAR1) Field Descriptions | 17-52 |
| 17-48 | 64-Bit Low Memory Base Address Register Field Descriptions | 17-53 |
| 17-49 | Bit Setting for 64-Bit High Memory Base Address Register | 17-53 |
| 17-50 | PCI Express Subsystem Vendor ID Register Field Description | 17-54 |
| 17-51 | PCI Express Subsystem ID Register Field Description | 17-54 |
| 17-52 | Capabilities Pointer Register Field Description | 17-55 |
| 17-53 | PCI Express Interrupt Line Register Field Description | 17-55 |
| 17-54 | PCI Express Interrupt Pin Register Field Description | 17-56 |
| 17-55 | PCI Express Maximum Grant Register Field Description | 17-56 |
| 17-56 | PCI Express Maximum Latency Register Field Description | 17-57 |
| 17-57 | PEXCSRBAR Field Descriptions | 17-58 |
| 17-58 | PCI Express Primary Bus Number Register Field Description | 17-58 |
| 17-59 | PCI Express Secondary Bus Number Register Field Description | 17-59 |
| 17-60 | PCI Express Subordinate Bus Number Register Field Description | 17-59 |
| 17-61 | PCI Express I/O Base Register Field Description | 17-60 |
| 17-62 | PCI Express I/O Limit Register Field Description | 17-61 |
| 17-63 | PCI Express Secondary Status Register Field Description | 17-61 |
| 17-64 | PCI Express Memory Base Register Field Description | 17-62 |
| 17-65 | PCI Express Memory Limit Register Field Description | 17-62 |
| 17-66 | PCI Express Prefetchable Memory Base Register Field Description | 17-63 |
| 17-67 | PCI Express Prefetchable Memory Limit Register Field Description | 17-63 |
| 17-68 | PCI Express Prefetchable Base Upper 32 Bits Register | 17-64 |
| 17-69 | PCI Express Prefetchable Limit Upper 32 Bits Register | 17-64 |
| 17-70 | PCI Express I/O Base Upper 16 Bits Register Field Description | 17-65 |
| 17-71 | PCI Express I/O Limit Upper 16 Bits Register Field Description | 17-65 |
| 17-72 | Capabilities Pointer Register Field Description | 17-65 |
| 17-73 | PCI Express Interrupt Line Register Field Description | 17-66 |
| 17-74 | PCI Express Interrupt Pin Register Field Description | 17-66 |
| 17-75 | PCI Express Bridge Control Register Field Description | 17-67 |

# Tables

| Table Number | Title | Page Number |
|---|---|---|

| 17-76 | PCI Express Power Management Capability ID Register Field Description | 17-69 |
| 17-77 | PCI Express Power Management Capabilities Register Field Description | 17-69 |
| 17-78 | PCI Express Status and Control Register Field Description | 17-70 |
| 17-79 | PCI Express Power Management Data Register Field Description | 17-70 |
| 17-80 | PCI Express Capability ID Register Field Description | 17-71 |
| 17-81 | PCI Express Capabilities Register Field Description | 17-71 |
| 17-82 | PCI Express Device Capabilities Register Field Description | 17-72 |
| 17-83 | PCI Express Device Control Register Field Description | 17-73 |
| 17-84 | PCI Express Device Status Register Field Description | 17-73 |
| 17-85 | PCI Express Link Capabilities Register Field Description | 17-74 |
| 17-86 | PCI Express Link Control Register Field Description | 17-74 |
| 17-87 | PCI Express Link Status Register Field Description | 17-75 |
| 17-88 | PCI Express Slot Capabilities Register Field Description | 17-76 |
| 17-89 | PCI Express Slot Control Register Field Description | 17-77 |
| 17-90 | PCI Express Slot Status Register Field Descriptions | 17-77 |
| 17-91 | PCI Express Root Control Register Field Description | 17-78 |
| 17-92 | PCI Express Root Status Register Field Description | 17-79 |
| 17-93 | PCI Express Capability ID Register Field Description | 17-79 |
| 17-94 | PCI Express MSI Message Control Register Field Description | 17-80 |
| 17-95 | PCI Express MSI Message Address Register Field Description | 17-80 |
| 17-96 | PCI Express MSI Message Upper Address Register Field Description | 17-81 |
| 17-97 | PCI Express MSI Message Data Register Field Description | 17-81 |
| 17-98 | PCI Express Advanced Error Reporting Capability ID Register Field Description | 17-83 |
| 17-99 | PCI Express Uncorrectable Error Status Register Field Description | 17-83 |
| 17-100 | PCI Express Uncorrectable Error Mask Register Field Description | 17-84 |
| 17-101 | PCI Express Uncorrectable Error Severity Register Field Description | 17-85 |
| 17-102 | PCI Express Correctable Error Status Register Field Description | 17-86 |
| 17-103 | PCI Express Correctable Error Mask Register Field Description | 17-87 |
| 17-104 | PCI Express Advanced Error Capabilities and Control Register Field Description | 17-87 |
| 17-105 | PCI Express Header Log Register Field Description | 17-88 |
| 17-106 | PCI Express Root Error Command Register Field Description | 17-89 |
| 17-107 | PCI Express Root Error Command Status Field Description | 17-89 |
| 17-108 | PCI Express Correctable Error Source ID Register Field Description | 17-90 |
| 17-109 | PCI Express Correctable Error Source ID Register Field Description | 17-90 |
| 17-110 | PEX_LTSSM_STAT Field Descriptions | 17-91 |
| 17-111 | PEX_LTSSM_STAT Status Codes | 17-91 |
| 17-112 | PEX_GCLK_RATIO Field Descriptions | 17-93 |
| 17-113 | PEX_PM_TIMER Field Descriptions | 17-94 |
| 17-114 | PEX_PME_TIMEOUT Field Descriptions | 17-94 |
| 17-115 | PEX_SSVID_UPDATE Field Descriptions | 17-95 |
| 17-116 | PEX_CFG_READY Field Descriptions | 17-96 |

# Tables

**Table Number** **Title** **Page Number**

17-117 PEX_PME_TO_ACK_TOR Field Descriptions 17-96
17-118 PEX_SS_INTR_MASK Field Descriptions 17-97
17-119 PCI Express Transactions 17-99
17-120 Lane Assignment With and Without Lane Reversal 17-101
17-121 Internal Platform (OCeaN) Message Data Format 17-103
17-122 PCI Express ATMU Outbound Messages 17-104
17-123 PCI Express RC Inbound Message Handling 17-105
17-124 PCI Express EP Inbound Message Handling 17-106
17-125 PCI Express Internal Controller Interrupt Sources 17-109
17-126 Error Conditions 17-111
17-127 Initial credit advertisement 17-114
17-128 Power Management State Supported 17-115
18-1 Signal Properties 18-4
18-2 Detailed Signal Descriptions 18-4
18-3 eSPI Registers 18-5
18-4 SPMODE Field Descriptions 18-6
18-5 SPIE Field Descriptions 18-7
18-6 SPIM Field Descriptions 18-8
18-7 SPCOM Field Descriptions 18-9
18-8 SPMODE*n* Field Descriptions 18-12
19-1 SATA Register Summary 19-4
19-2 CQR Field Descriptions 19-6
19-3 CAR Field Descriptions 19-6
19-4 CCR Field Descriptions 19-7
19-5 CER Field Descriptions 19-8
19-6 DER Field Descriptions 19-8
19-7 CHBA Field Descriptions 19-9
19-8 HStatus Field Descriptions 19-9
19-9 HControl Field Descriptions 19-12
19-10 CQPMP Field Descriptions 19-13
19-11 SIG Register Field Descriptions 19-14
19-12 ICC Field Descriptions 19-15
19-13 SStatus Field Descriptions 19-15
19-14 SError Field Descriptions 19-16
19-15 SControl Field Descriptions 19-19
19-16 SNotification Field Descriptions 19-20
19-17 TransCfg Field Descriptions 19-20
19-18 TransStatus Field Descriptions 19-21
19-19 LinkCfg Field Descriptions 19-21
19-20 LinkCfg1 Field Descriptions 19-23
19-21 LinkCfg2 Field Descriptions 19-23

# Tables

Table Number | Title | Page Number

19-22 LinkStatus Field Descriptions ... 19-24
19-23 LinkStatus1 Field Descriptions ... 19-25
19-24 PhyCtrlCfg1 Field Descriptions ... 19-26
19-25 CommandStatus Field Descriptions ... 19-27
19-26 SYSPR Field Descriptions ... 19-29
19-27 Word 0—Data Base Address ... 19-30
19-28 Word 1—FIS_LEN ... 19-30
19-29 Word 2—Data Base Address ... 19-30
19-30 Word 3—Description Information ... 19-30
19-31 Word 0—Data Base Address ... 19-33
19-32 Word 3—Description Information ... 19-33
19-33 Vendor BIST Test—Command Header ... 19-34
19-34 Vendor BIST Test—Command Descriptor ... 19-34
19-35 Read DMA Command—Command Header ... 19-42
19-36 Read DMA Command—Command Descriptor ... 19-42
19-37 Read DMA Command—PRD Entries ... 19-42
20-1 Signal Properties ... 20-4
20-2 eSDHC Memory Map ... 20-5
20-3 DSADDR Field Descriptions ... 20-6
20-4 BLKATTR Field Descriptions ... 20-7
20-5 CMDARG Field Descriptions ... 20-7
20-6 XFERTYP Field Descriptions ... 20-8
20-7 Determination of Transfer Type ... 20-10
20-8 Relation Between Parameters and Name of Response Type ... 20-10
20-9 Response Bit Definition for Each Response Type ... 20-11
20-10 DATPORT Field Descriptions ... 20-12
20-11 PRSSTAT Field Descriptions ... 20-13
20-12 PROCTL Field Descriptions ... 20-17
20-13 SYSCTL Field Descriptions ... 20-20
20-14 IRQSTAT Field Descriptions ... 20-23
20-15 Relation Between Command Timeout Error and Command Complete Status ... 20-26
20-16 Relation Between Data Timeout Error and Transfer Complete Status ... 20-26
20-17 Relation Between Command CRC Error and Command Timeout Error ... 20-26
20-18 IRQSTATEN Field Descriptions ... 20-27
20-19 IRQSIGEN Field Descriptions ... 20-29
20-20 AUTOC12ERR Field Descriptions ... 20-31
20-21 Relationship Between Command CRC Error and Command Timeout Error for Auto CMD12 ... 20-32
20-22 HOSTCAPBLT Field Descriptions ... 20-33
20-23 WML Field Descriptions ... 20-34
20-24 FEVT Field Descriptions ... 20-35

# Tables

| Table Number | Title | Page Number |
|---|---|---|
| 20-25 | HOSTVER Field Descriptions | 20-36 |
| 20-26 | DCR Field Descriptions | 20-37 |
| 20-27 | Commands for MMC/SD | 20-55 |
| 20-28 | EXT_CSD Access Modes | 20-60 |
| 21-1 | ULPI Signal Descriptions | 21-3 |
| 21-2 | USB Interface Memory Map | 21-4 |
| 21-3 | CAPLENGTH Register Field Descriptions | 21-7 |
| 21-4 | HCIVERSION Register Field Descriptions | 21-7 |
| 21-5 | HCSPARAMS Register Field Descriptions | 21-8 |
| 21-6 | HCCPARAMS Register Field Descriptions | 21-9 |
| 21-7 | DCIVERSION Register Field Descriptions | 21-10 |
| 21-8 | DCCPARAMS Register Field Descriptions | 21-10 |
| 21-9 | USBCMD Register Field Descriptions | 21-11 |
| 21-10 | USBSTS Register Field Descriptions | 21-14 |
| 21-11 | USBINTR Register Field Descriptions | 21-16 |
| 21-12 | FRINDEX Register Field Descriptions | 21-18 |
| 21-13 | FRINDEX N Values | 21-18 |
| 21-14 | PERIODICLISTBASE Register Field Descriptions | 21-19 |
| 21-15 | DEVICEADDR Register Field Descriptions | 21-19 |
| 21-16 | ASYNCLISTADDR Register Field Descriptions | 21-20 |
| 21-17 | ENDPOINTLISTADDR Register Field Descriptions | 21-20 |
| 21-18 | BURSTSIZE Register Field Descriptions | 21-21 |
| 21-19 | TXFILLTUNING Register Field Descriptions | 21-22 |
| 21-20 | ULPI VIEWPORT Field Descriptions | 21-23 |
| 21-21 | CONFIGFLAG Register Field Descriptions | 21-24 |
| 21-22 | PORTSC Register Field Descriptions | 21-25 |
| 21-23 | USBMODE Register Field Descriptions | 21-30 |
| 21-24 | ENDPTSETUPSTAT Register Field Descriptions | 21-31 |
| 21-25 | ENDPTPRIME Register Field Descriptions | 21-31 |
| 21-26 | ENDPTFLUSH Register Field Descriptions | 21-32 |
| 21-27 | ENDPTSTATUS Register Field Descriptions | 21-32 |
| 21-28 | ENDPTCOMPLETE Register Field Descriptions | 21-33 |
| 21-29 | ENDPTCTRL0 Register Field Descriptions | 21-34 |
| 21-30 | ENDPTCTRL*x* Register Field Descriptions | 21-35 |
| 21-31 | SNOOP*n* Register Field Descriptions | 21-37 |
| 21-32 | AGE_CNT_THRESH Register Field Descriptions | 21-38 |
| 21-33 | PRI_CTRL Register Field Descriptions | 21-38 |
| 21-34 | SI_CTRL Register Field Descriptions | 21-39 |
| 21-35 | CONTROL Field Descriptions | 21-40 |
| 21-36 | Supported PHY Interfaces | 21-41 |
| 21-37 | Typ Field Encodings | 21-43 |

# Tables

| Table Number | Title | Page Number |
|---|---|---|
| 21-38 | Next Schedule Element Pointer | 21-45 |
| 21-39 | iTD Transaction Status and Control | 21-46 |
| 21-40 | Buffer Pointer Page 0 (Plus) | 21-46 |
| 21-41 | iTD Buffer Pointer Page 1 (Plus) | 21-47 |
| 21-42 | Buffer Pointer Page 2 (Plus) | 21-47 |
| 21-43 | Buffer Pointer Page 3–6 | 21-47 |
| 21-44 | Next Link Pointer | 21-48 |
| 21-45 | Endpoint and Transaction Translator Characteristics | 21-48 |
| 21-46 | Micro-Frame Schedule Control | 21-49 |
| 21-47 | siTD Transfer Status and Control | 21-49 |
| 21-48 | siTD Buffer Pointer Page 0 (Plus) | 21-50 |
| 21-49 | siTD Buffer Pointer Page 1 (Plus) | 21-51 |
| 21-50 | siTD Back Link Pointer | 21-51 |
| 21-51 | qTD Next Element Transfer Pointer (DWord 0) | 21-52 |
| 21-52 | qTD Alternate Next Element Transfer Pointer (DWord 1) | 21-52 |
| 21-53 | qTD Token (DWord 2) | 21-53 |
| 21-54 | qTD Buffer Pointer | 21-56 |
| 21-55 | Queue Head DWord 0 | 21-57 |
| 21-56 | Endpoint Characteristics: Queue Head DWord 1 | 21-58 |
| 21-57 | Endpoint Capabilities: Queue Head DWord 2 | 21-58 |
| 21-58 | Current qTD Link Pointer | 21-59 |
| 21-59 | Host-Controller Rules for Bits in Overlay (DWords 5, 6, 8, and 9) | 21-60 |
| 21-60 | FTSN Normal Path Pointer | 21-61 |
| 21-61 | FSTN Back Path Link Pointer | 21-61 |
| 21-62 | Default Values of Operational Register Space | 21-62 |
| 21-63 | Behavior During Wake-up Events | 21-65 |
| 21-64 | Operation of FRINDEX and SOFV (SOF Value Register) | 21-69 |
| 21-65 | Example Periodic Reference Patterns for Interrupt Transfers | 21-82 |
| 21-66 | Ping Control State Transition Table | 21-83 |
| 21-67 | Interrupt IN/OUT Do Complete Split State Execution Criteria | 21-97 |
| 21-68 | Initial Conditions for OUT siTD TP and T-Count Fields | 21-105 |
| 21-69 | Transaction Position (TP)/Transaction Count (T-Count) Transition Table | 21-105 |
| 21-70 | Summary siTD Split Transaction State | 21-109 |
| 21-71 | Example Case 2a—Software Scheduling siTDs for an IN Endpoint | 21-110 |
| 21-72 | Summary of Transaction Errors | 21-113 |
| 21-73 | Summary Behavior on Host System Errors | 21-116 |
| 21-74 | Endpoint Capabilities/Characteristics | 21-118 |
| 21-75 | Current dTD Pointer | 21-119 |
| 21-76 | Multiple Mode Control | 21-120 |
| 21-77 | Next dTD Pointer | 21-120 |
| 21-78 | dTD Token | 21-121 |

# Tables

| Table Number | Title | Page Number |
|---|---|---|
| 21-79 | Buffer Pointer Page 0 | 21-121 |
| 21-80 | Buffer Pointer Page 1 | 21-122 |
| 21-81 | Buffer Pointer Pages 2–4 | 21-122 |
| 21-82 | Device Controller State Information Bits | 21-124 |
| 21-83 | Device Controller Endpoint Initialization | 21-127 |
| 21-84 | Device Controller Stall Response Matrix | 21-128 |
| 21-85 | Variable Length Transfer Protocol Example (ZLT = 0) | 21-130 |
| 21-86 | Variable Length Transfer Protocol Example (ZLT = 1) | 21-130 |
| 21-87 | Interrupt/Bulk Endpoint Bus Response Matrix | 21-131 |
| 21-88 | Control Endpoint Bus Response Matrix | 21-133 |
| 21-89 | Isochronous Endpoint Bus Response Matrix | 21-136 |
| 21-90 | Device Error Matrix | 21-140 |
| 21-91 | Error Descriptions | 21-141 |
| 21-92 | Interrupt Handling Order | 21-141 |
| 21-93 | Low Frequency Interrupt Events | 21-142 |
| 21-94 | Error Interrupt Events | 21-142 |
| 21-95 | Functional Differences Between EHCI and EHCI with Embedded TT | 21-143 |
| 21-96 | Emulated Handshakes | 21-145 |
| 21-97 | ULPI Timing | 21-148 |
| 22-1 | IPIC External Signals—Detailed Signal Descriptions | 22-2 |
| 22-2 | GPIO Register Address Map | 22-2 |
| 22-3 | GPDIR Bit Settings | 22-3 |
| 22-4 | GPODR Bit Settings | 22-3 |
| 22-5 | GP*n*DAT Bit Settings | 22-4 |
| 22-6 | GPIER Bit Settings | 22-4 |
| 22-7 | GPIMR Bit Settings | 22-5 |
| 22-8 | GPICR Bit Settings | 22-5 |
| 23-1 | External Signal Summary | 23-2 |
| 23-2 | Detailed Signal Descriptions | 23-2 |
| 23-3 | Global Utilities Module Memory Map | 23-4 |
| 23-4 | PORPLLSR Field Descriptions | 23-6 |
| 23-5 | PORBMSR Field Descriptions | 23-7 |
| 23-6 | PORIMPSCR Field Descriptions | 23-8 |
| 23-7 | PORDEVSR Field Descriptions | 23-9 |
| 23-8 | PORDBGMSR Field Descriptions | 23-12 |
| 23-9 | PORDEVSR2 Field Descriptions | 23-13 |
| 23-10 | GPPORCR Field Descriptions | 23-14 |
| 23-11 | GENCFGR Field Descriptions | 23-14 |
| 23-12 | PMUXCR Field Descriptions | 23-15 |
| 23-13 | DEVDISR Field Descriptions | 23-17 |
| 23-14 | PMJCR Field Descriptions | 23-20 |

# Tables

| Table Number | Title | Page Number |
|---|---|---|
| 23-15 | POWMGTCSR Field Descriptions | 23-21 |
| 23-16 | PMRCCR Field Descriptions | 23-23 |
| 23-17 | PMPDCCR Register Field Descriptions | 23-25 |
| 23-18 | PMCDR Register Field Descriptions | 23-26 |
| 23-19 | MCPSUMR Field Descriptions | 23-27 |
| 23-20 | RSTRSCR Field Descriptions | 23-28 |
| 23-21 | ECTRSTCR Field Descriptions | 23-28 |
| 23-22 | AUTORSTSR Field Descriptions | 23-29 |
| 23-23 | PVR Field Descriptions | 23-30 |
| 23-24 | SVR Field Descriptions | 23-30 |
| 23-25 | RSTCR Field Descriptions | 23-31 |
| 23-26 | LBCVSELCR Field Descriptions | 23-31 |
| 23-27 | DDRCLKDR Field Descriptions | 23-32 |
| 23-28 | CLKOCR Field Descriptions | 23-33 |
| 23-29 | ECMCR Field Descriptions | 23-34 |
| 23-30 | GCR Field Descriptions | 23-34 |
| 23-31 | SRDS1CR0 Field Descriptions | 23-36 |
| 23-32 | SRDS1CR2 Field Descriptions | 23-37 |
| 23-33 | SRDS2CR0 Field Descriptions | 23-40 |
| 23-34 | SRDS2CR1 Field Descriptions | 23-41 |
| 23-35 | SRDS2CR2 Field Descriptions | 23-43 |
| 23-36 | SRDS2CR3 Field Descriptions | 23-45 |
| 23-37 | Power Management Entry Protocol and Initiating Functional Units | 23-54 |
| 24-1 | Control Register Memory Map | 24-4 |
| 24-2 | PMGC0 Field Descriptions | 24-5 |
| 24-3 | PMLCA0 Field Descriptions | 24-6 |
| 24-4 | PMLCA1–PMLCA9 Field Descriptions | 24-7 |
| 24-5 | PMLCB0 Field Descriptions | 24-8 |
| 24-6 | PMLCB*n* Field Descriptions | 24-9 |
| 24-7 | PMC0 Field Descriptions | 24-11 |
| 24-8 | PMC[1–9] Field Descriptions | 24-11 |
| 24-9 | Burst Definition | 24-14 |
| 24-10 | Performance Monitor Events | 24-17 |
| 24-11 | PMGC0 and PMLCAn Settings | 24-27 |
| 24-12 | Register Settings for Counting Examples | 24-28 |
| 25-1 | POR Configuration Settings and Debug Modes | 25-3 |
| 25-2 | Debug, Watchpoint and Test Signal Summary | 25-5 |
| 25-3 | Debug Signals—Detailed Signal Descriptions | 25-6 |
| 25-4 | Watchpoint and Trigger Signals—Detailed Signal Descriptions | 25-7 |
| 25-5 | JTAG Test and Other Signals—Detailed Signal Descriptions | 25-8 |
| 25-6 | Debug and Watchpoint Monitor Memory Map | 25-9 |

# Tables

Table Number | Title | Page Number

25-7 WMCR0 Field Descriptions.......... 25-11
25-8 WMCR1 Field Descriptions.......... 25-12
25-9 WMAR Field Descriptions.......... 25-13
25-10 WMAMR Field Descriptions.......... 25-13
25-11 WMTMR Field Descriptions.......... 25-14
25-12 Transaction Types By Interface.......... 25-14
25-13 WMSR Field Descriptions.......... 25-15
25-14 TBCR0 Field Descriptions.......... 25-16
25-15 TBCR1 Field Descriptions.......... 25-17
25-16 TBAR Field Descriptions.......... 25-18
25-17 TBAMR Field Descriptions.......... 25-18
25-18 TBTMR Field Descriptions.......... 25-19
25-19 TBSR Field Descriptions.......... 25-20
25-20 TBACR Field Descriptions.......... 25-21
25-21 TBADHR Field Descriptions.......... 25-21
25-22 TBADR Field Descriptions.......... 25-22
25-23 PCIDR Field Descriptions.......... 25-22
25-24 CCIDR Field Descriptions.......... 25-23
25-25 TOSR Field Descriptions.......... 25-24
25-26 Source and Target ID Values.......... 25-24
25-27 CMD Trace Buffer Entry Field Descriptions (TBCR1[IFSEL] = 000).......... 25-27
25-28 DDR Trace Buffer Entry Field Descriptions (TBCR1[IFSEL] = 001).......... 25-28
25-29 PCI Trace Buffer Entry Field Descriptions (TBCR1[IFSEL] = 010).......... 25-29
25-30 PCI Express Trace Buffer Entry Field Descriptions (TBCR1[IFSEL] = 100 or 101 or 110).......... 25-29
A-1 Local Configuration Control Registers.......... A-1
A-2 Local Access Window Registers.......... A-1
A-3 ECM Registers.......... A-2
A-4 DDR Memory Controller Registers.......... A-3
A-5 I$^2$C Controller 1 & 2 Registers.......... A-5
A-6 DUARTRegisters.......... A-5
A-7 Enhanced Local Bus Controller Registers.......... A-6
A-8 Serial Peripheral Interface Registers.......... A-8
A-9 PCI Controller Registers.......... A-8
A-10 PCI Express Controller 1 & 2 Registers..........A-11
A-11 GPIO Registers.......... A-15
A-12 SATA Registers.......... A-16
A-13 L2/SRAM Memory-Mapped Registers.......... A-17
0-1 USB Interface Memory Map.......... A-18
A-2 Module Memory Map.......... A-20

# Tables

Table Number | Title | Page Number

1-3 eSDHC Memory Map ..... A-30
1-4 SEC Address Map ..... A-31
A-14 PIC Global Registers ..... A-37
A-15 PIC Interrupt Source Registers ..... A-41
A-16 PIC Processor (per-CPU) Registers ..... A-47
A-17 Global Utilities Registers ..... A-48
A-18 Performance Monitor Registers ..... A-50
A-19 Watchpoint Monitor and Trace Buffer Registers ..... A-51
C-1 Comparison of Features, MPC8536E and MPC8535E ..... C-1
C-2 Boot ROM Location (MPC8535E) ..... C-3
C-3 Host/Agent Configuration (MPC8535E) ..... C-4
C-4 SerDes1 I/O Port Selection (MPC8535E) ..... C-4
C-5 SerDes2 I/O Port Selection (MPC8535E) ..... C-5
C-6 Supported SerDes 1 (PCI Express) Configurations ..... C-5

# Tables

| Table<br>Number | Title | Page<br>Number |
|---|---|---|

# About This Book

This reference manual defines the functionality of the MPC8536E. This device integrates a PowerPC™ processor core with system logic required for networking, telecommunications, and wireless infrastructure applications. The e500v2 processor core is a low-power implementation of the family of reduced instruction set computing (RISC) embedded processors that implement the Book E definition of the PowerPC architecture. This book is intended as a companion to the *PowerPC e500 Core Complex Reference Manual*.

# Audience

It is assumed that the reader understands operating systems, microprocessor system design, and the basic principles of RISC processing.

# Organizations

Following is a summary and a brief description of the major parts of this reference manual:

Part I, “Overview,” describes the many features of the MPC8536E integrated host processor at an overview level. The following chapters are included:

- Chapter 1, “Overview,” provides a high-level description of features and functionality of the MPC8536E integrated host processor. It describes the MPC8536E, its interfaces, and its programming model. The functional operation of the MPC8536E with emphasis on peripheral functions is also described.
- Chapter 2, “Memory Map,” describes the memory map of the MPC8536E. An overview of the local address map is followed by a description of how local access windows are used to define the local address map. The inbound and outbound address translation mechanisms used to map to and from external memory spaces are described next. Finally, the configuration, control, and status registers are described, including a complete listing of all memory-mapped registers with cross references to the sections detailing descriptions of each.
- Chapter 3, “Signal Descriptions,” provides a listing of all the external signals, cross-references for signals that serve multiple functions, output signal states at reset, and reset configuration signals (and the modes they define).
- Chapter 4, “Reset, Clocking, and Initialization,” describes the hard and soft resets, the power-on reset (POR) sequence, power-on reset configuration, clocking, and initialization of the MPC8536E.

Part II, “e500 Core Complex and L2 Cache,” describes the integration of the e500v2 core in the MPC8536E and the interaction between the core complex and the L2 cache. The following chapters are included:

- Chapter 5, “e500 Core Integration Details,” provides an overview of the e500v2 core processor and how it is implemented in the MPC8536E.
- Chapter 6, “L2 Look-Aside Cache/SRAM,” describes the L2 cache of the MPC8536E. Note that the L2 cache can also be addressed directly as memory-mapped SRAM.

Part III, “Memory, Security, and I/O Interfaces,” defines the memory, security, and I/O interfaces of the MPC8536E and how these blocks interact with one another and with other blocks on the device. The following chapters are included:

- Chapter 7, “e500 Coherency Module,” defines the e500v2 coherency module and how it facilitates communication between the e500v2 core complex, the L2 cache, and the other blocks that comprise the coherent memory domain of the MPC8536E.

  The ECM provides a mechanism for I/O-initiated transactions to snoop the core complex bus (CCB) of the e500v2 core in order to maintain coherency across cacheable local memory. It also provides a flexible, easily expandable switch-type structure for e500v2- and I/O-initiated transactions to be routed (dispatched) to target modules on the MPC8536E.
- Chapter 8, “DDR Memory Controller,” describes the two DDR2/DDR3 SDRAM memory controllers of the MPC8536E. These fully programmable controllers support most DDR memories available today, including both buffered and unbuffered devices. The built-in error checking and correction (ECC) ensures very low bit-error rates for reliable high-frequency operation. Dynamic power management and auto-precharge modes simplify memory system design. A large set of special features like crawl mode and ECC error injection support rapid system debug.
- Chapter 9, “Programmable Interrupt Controller (PIC),” describes the embedded programmable interrupt controller (PIC) of the MPC8536E. The PIC is an OpenPIC-compliant interrupt controller that provides interrupt management and is responsible for receiving hardware-generated interrupts from different sources (both internal and external), prioritizing them and delivering them to the CPU for servicing.
- Chapter 10, “Security Engine (SEC) 3.0,” describes the security controller of the MPC8536E. The SEC 3.0 off-loads computationally intensive security functions, such as key generation and exchange, authentication, and bulk encryption from the processor cores of the MPC8536E. It is optimized to process all cryptographic algorithms associated with IPsec, IKE, SSL/TLS, iSCSI, SRTP, 802.11i, 3G, A5/3 for GSM and EDGE, and GEA3 for GPRS.
- Chapter 11, “I$^2$C Interfaces,” describes the inter-IC (IIC or I$^2$C) bus controllers of the MPC8536E. This synchronous, serial, bidirectional, multi-master bus allows two-wire connection of devices, such as microcontrollers, EEPROMs, real-time clock devices, A/D converters and LCDs. The MPC8536E powers up in boot sequencer mode which allows the I$^2$C1 controller to initialize configuration registers.
- Chapter 12, “DUART,” describes the (dual) universal asynchronous receiver/transmitters (UARTs) which feature a PC16552D-compatible programming model. These independent UARTs are provided specifically to support system debugging.

- Chapter 13, “Enhanced Local Bus Controller,” describes the enhanced local bus controller (eLBC) of the MPC8536E. The main component of the enhanced local bus controller is its memory controller which provides a seamless interface to many types of memory devices and peripherals. The memory controller is responsible for controlling eight memory banks shared by a general-purpose chip-select machine (GPCM), a NAND flash control machine (FCM), and up to three user-programmable machines (UPMs). As such, it supports a minimal glue logic interface to SRAM, EPROM, Flash EPROM, burstable RAM, regular DRAM devices, extended data output DRAM devices, and other peripherals.
- Chapter 14, “Enhanced Three-Speed Ethernet Controllers,” describes the two enhanced three-speed Ethernet controllers on the MPC8536E. These controllers provide 10/100/1Gb Ethernet support with a complete set of media-independent interface options including MII, RMII, GMII, RGMII, TBI, and RTBI. Each controller provides very high throughput using a captive DMA channel and direct connection to the MPC8536E memory coherency module. The controllers provide two full-duplex FIFO interface modes and quality of service support. They are backward compatible with PowerQUICC III TSEC controllers.
- Chapter 15, “DMA Controller,” describes the four-channel general-purpose DMA controller of the MPC8536E. The DMA controller transfers blocks of data independent of the e500v2 core or external hosts. Data movement occurs among the local address space. The DMA controller has four high-speed channels. Both the e500 core and external masters can initiate a DMA transfer. All channels are capable of complex data movement and advanced transaction chaining.
- Chapter 16, “PCI Bus Interface,” describes the PCI interface, which complies with the *PCI Local Bus Specification*, Rev. 2.3. This chapter provides a basic description of PCI bus operations. The specific emphasis is directed at how this device implements the PCI specification.
- Chapter 17, “PCI Express Interface Controller,” describes the three PCI Express® controllers of the MPC8536E. Each controller is compliant with the *PCI Express Base Specification Revision 1.0a*. The physical layer of these controllers operate at 2.5 Gbaud per lane. Configuration options allow multiple width configurations among the three controllers.
- Chapter 18, “Enhanced Serial Peripheral Interface,” describes the MPC8536E enhanced serial peripheral interface (SPI) that allows the exchange of data between MPC85xx family devices. The SPI can also be used to communicate with peripheral devices such as EEPROMs, real-time clocks, A/D converters, and ISDN devices.
- Chapter 19, “SATA Controller,” describes the serial ATA controllers of the MPC8536E.
- Chapter 20, “Enhanced Secure Digital Host Controller,” describes the enhanced SD host controller, which provides an interface between the host system and SD and MMC cards. It provides a functional description of the major system blocks and includes command information for the host.
- Chapter 21, “Universal Serial Bus Interfaces,” describes the three universal serial bus (USB) interfaces. The USB module is a USB 2.0-compliant serial interface engine for implementing a USB interface. The registers and data structures are based on the *Enhanced Host Controller Interface Specification for Universal Serial Bus* (EHCI) from Intel Corporation. The USB dual-role modules can act as a host or as a device on the USB bus.
- Chapter 22, “General Purpose I/O (GPIO),” describes the general-purpose input and output signals of the MPC8536E.

Part IV, “Global Functions and Debug,” defines other global blocks of the MPC8536E. The following chapters are included:

- Chapter 23, “Global Utilities,” defines the global utilities of the MPC8536E. These include power management, I/O device enabling, power-on-reset (POR) configuration monitoring, general-purpose I/O signal use, and multiplexing for the interrupt and local bus chip select signals.
- Chapter 24, “Device Performance Monitor,” describes the performance monitor of the MPC8536E. Note that the MPC8536E performance monitor is similar to but separate from the performance monitor implemented on the e500v2 core.
- Chapter 25, “Debug Features and Watchpoint Facility,” describes the debug features and watchpoint monitor of the MPC8536E.

This reference manual also contains the following appendices:

- Appendix A, “Complete List of Configuration, Control, and Status Registers,” lists all memory-mapped registers by block.
- Appendix B, “Revision History,” contains a list of major differences since the last revision of the document.
- Appendix C, “MPC8535E,” details differences between the MPC8536E and MPC8535E devices.

It also contains a glossary and an index.

# Suggested Reading

This section lists additional reading that provides background for the information in this manual as well as general information about the architecture.

## General Information

The following documentation, published by Morgan-Kaufmann Publishers, 340 Pine Street, Sixth Floor, San Francisco, CA, provides useful information about the PowerPC architecture and computer architecture in general:

- *The PowerPC Architecture: A Specification for a New Family of RISC Processors*, Second Edition, by International Business Machines, Inc.
- *Computer Architecture: A Quantitative Approach*, Third Edition, by John L. Hennessy and David A. Patterson
- *Computer Organization and Design: The Hardware/Software Interface*, Second Edition, by David A. Patterson and John L. Hennessy

## Related Documentation

Freescale documentation is available from the sources listed on the back cover of this manual; the document order numbers are included in parentheses for ease in ordering:

- *EREF: A Reference for Freescale Book E and the e500 Core* (EREF)—This book provides a higher-level view of the programming model as it is defined by Book E, the Freescale Book E implementation standards, and the e500 microprocessor.

- *PowerPC™ e500 Core Family Reference Manual* (E500CORERM)—This book provides detailed information about the functions and features of the e500 v1 and v2 cores.
- Reference manuals (formerly called user’s manuals)—These books provide details about individual implementations.
- Addenda/errata to reference or user’s manuals—Because some processors have follow-on parts an addendum is provided that describes the additional features and functionality changes. These addenda are intended for use with the corresponding reference or user’s manuals.
- Hardware specifications—Hardware specifications provide specific data regarding bus timing, signal behavior, and AC, DC, and thermal characteristics, as well as other design considerations.
- Product Briefs—Each device has a technical summary that provides an overview of its features.
- Application notes—These short documents address specific design issues useful to programmers and engineers working with Freescale processors.

Additional literature is published as new processors become available. For a current list of documentation, refer to http://www.freescale.com.

# Conventions

This document uses the following notational conventions:

cleared/set    When a bit takes the value zero, it is said to be cleared; when it takes a value of one, it is said to be set.

**mnemonics**    Instruction mnemonics are shown in lowercase bold.

*italics*    Italics indicate variable command parameters, for example, **bcctr***x*.

Book titles in text are set in italics

Internal signals are set in lowercase italics, for example, $\overline{core\_int}$

0x0    Prefix to denote hexadecimal number

0b0    Prefix to denote binary number

**r**A, **r**B    Instruction syntax used to identify a source GPR

**r**D    Instruction syntax used to identify a destination GPR

REG[FIELD]    Abbreviations for registers are shown in uppercase text. Specific bits, fields, or ranges appear in brackets. For example, MSR[LE] refers to the little-endian mode enable bit in the machine state register.

x    In some contexts, such as signal encodings, an unitalicized x indicates a don’t care.

*x*    An italicized *x* indicates an alphanumeric variable.

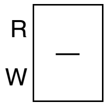
*n*    An italicized *n* indicates a numeric variable.

¬    NOT logical operator

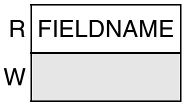
&    AND logical operator

|    OR logical operator

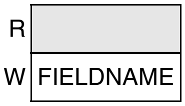
|| Concatenation, for example TCR[WP]||TCR[WPEXT]



Indicates a reserved bit field in a memory-mapped or an e500 register.



Indicates a read-only bit field in a memory-mapped register.



Indicates a write-only bit field in a memory-mapped register. Although these bits can be written to as ones or zeros, they are always read as zeros.

## Signal Conventions

$\overline{\text{OVERBAR}}$ An overbar indicates that a signal is active-low.

*lowercase_italics* Lowercase italics is used to indicate internal signals.

lowercase_plaintext Lowercase plain text is used to indicate signals that are used for configuration. For more information, see Section 3.2, “Configuration Signals Sampled at Reset.”

## Register Access Conventions

In the register figures and field descriptions, the following access definitions apply:

- Reserved fields are always ignored for the purposes of determining access type.
- R/W, R, and W (read/write, read only, and write only) indicate that all the non-reserved fields in a register have the same access type.
- w1c indicates that all of the non-reserved fields in a register are cleared by writing ones to them.
- Mixed indicates a combination of access types.
- Special is used when no other category applies. In this case the register figure and field description table should be read carefully.

## Acronyms and Abbreviations

Table i contains acronyms and abbreviations used in this document.

**Table i. Acronyms and Abbreviated Terms**

| Term | Meaning |
|---|---|
| ADB | Allowable disconnect boundary |
| ATMU | Address translation and mapping unit |
| BD | Buffer descriptor |
| BIST | Built-in self test |

Table i. Acronyms and Abbreviated Terms (continued)

| Term | Meaning |
|---|---|
| BTB | Branch target buffer |
| BUID | Bus unit ID |
| CAM | Content-addressable memory |
| CCB | Core complex bus |
| CCSR | Configuration control and status register |
| CEPT | Conférence des administrations européenes des postes et télécommunications (European Conference of Postal and Telecommunications Administrations) |
| COL | Collision |
| CRC | Cyclic redundancy check |
| CRS | Carrier sense |
| DDR | Double data rate |
| DMA | Direct memory access |
| DPLL | Digital phase-locked loop |
| DRAM | Dynamic random access memory |
| DUART | Dual universal asynchronous receiver/transmitter |
| EA | Effective address |
| ECC | Error checking and correction |
| ECM | e500 coherency module |
| EHPI | Enhanced host port interface |
| EPROM | Erasable programmable read-only memory |
| FCS | Frame-check sequence |
| GCI | General circuit interface |
| GMII | Gigabit media independent interface |
| GPCM | General-purpose chip-select machine |
| GPIO | General-purpose I/O |
| GPR | General-purpose register |
| GUI | Graphical user interface |
| $I^2C$ | Inter-integrated circuit |
| IDL | Inter-chip digital link |
| IEEE | Institute of Electrical and Electronics Engineers |
| IPG | Interpacket gap |
| ITLB | Instruction translation lookaside buffer |
| IU | Integer unit |
| JTAG | Joint Test Action Group |
| LAE | Local access error |

Table i. Acronyms and Abbreviated Terms (continued)

| Term | Meaning |
|---|---|
| LAW | Local access window |
| LBC | Local bus controller |
| LIFO | Last-in-first-out |
| LRU | Least recently used |
| LSB | Least-significant byte |
| lsb | Least-significant bit |
| LSU | Load/store unit |
| MAC | Multiply accumulate, media access control |
| MDI | Medium-dependent interface |
| MESI | Modified/exclusive/shared/invalid—cache coherency protocol |
| MII | Media independent interface |
| MMU | Memory management unit |
| MSB | Most-significant byte |
| msb | Most-significant bit |
| NMSI | Nonmultiplexed serial interface |
| No-op | No operation |
| OCeaN | On-chip network |
| OSI | Open systems interconnection |
| PCI | Peripheral component interconnect bus |
| PCMCIA | Personal Computer Memory Card International Association |
| PCS | Physical coding sublayer |
| PIC | Programmable interrupt controller |
| PMA | Physical medium attachment |
| PMD | Physical medium dependent |
| POR | Power-on reset |
| RGMII | Reduced gigabit media independent interface |
| RISC | Reduced instruction set computing |
| RTOS | Real-time operating system |
| RWITM | Read with intent to modify |
| RMW | Read modify write |
| Rx | Receive |
| RxBD | Receive buffer descriptor |
| SCC | Serial communication controller |
| SCP | Serial control port |
| SDLC | Synchronous data link control |

Table i. Acronyms and Abbreviated Terms (continued)

| Term | Meaning |
|---|---|
| SDMA | Serial DMA |
| SFD | Start frame delimiter |
| SI | Serial interface |
| SIU | System interface unit |
| SPR | Special-purpose register |
| SRAM | Static random access memory |
| TAP | Test access port |
| TBI | Ten-bit interface |
| TDM | Time-division multiplexed |
| TLB | Translation lookaside buffer |
| TSA | Time-slot assigner |
| TSEC | Three-speed Ethernet controller |
| Tx | Transmit |
| TxBD | Transmit buffer descriptor |
| UART | Universal asynchronous receiver/transmitter |
| UPM | User-programmable machine |
| UTP | Unshielded twisted pair |
| VA | Virtual address |
| ZBT | Zero bus turnaround |

# Part I
# Overview

Part I describes the many features of the MPC8536E integrated host processor at an overview level. The following chapters are included:

- Chapter 1, “Overview,” provides a high-level description of features and functionality of the MPC8536E integrated host processor. It describes the MPC8536E, its interfaces, and its programming model. The functional operation of the MPC8536E with emphasis on peripheral functions is also described.
- Chapter 2, “Memory Map,” describes the memory map of the MPC8536E. An overview of the local address map is followed by a description of how local access windows are used to define the local address map. The inbound and outbound address translation mechanisms used to map to and from external memory spaces are described next. Finally, the configuration, control, and status registers are described, including a complete listing of all memory-mapped registers with cross references to the sections detailing descriptions of each.
- Chapter 3, “Signal Descriptions,” provides a listing of all the external signals, cross-references for signals that serve multiple functions, output signal states at reset, and reset configuration signals (and the modes they define).
- Chapter 4, “Reset, Clocking, and Initialization,” describes the hard and soft resets, the power-on reset (POR) sequence, power-on reset configuration, clocking, and initialization of the MPC8536E.

# Chapter 1
# Overview

The MPC8536E integrates a PowerPC™ processor core with system logic required for imaging, networking, and communications applications. The MPC8536E is a member of the PowerQUICC III™ family of devices that combine system-level support for industry-standard interfaces with processors that implement the PowerPC architecture. This chapter provides a high-level description of features and functionality of the MPC8536E integrated processor.

Although this chapter is written from the perspective of the MPC8536E, most of the material applies to the MPC8535E as well. For information on differences between the MPC8535E and the MPC8536E, see Appendix C, "MPC8535E."

## 1.1 Introduction

The MPC8536E uses the e500 core and high-speed interconnect technology to balance processor performance with I/O system throughput. The e500 core implements the enhanced Book E instruction set architecture and provides unprecedented levels of hardware and software debugging support.

In addition, the MPC8536E offers a double-precision floating-point auxiliary processing unit (APU), 512Kbytes of level-2 cache, two integrated 10/100/1Gb enhanced three-speed Ethernet controllers (eTSECs) with TCP/IP acceleration and classification capabilities and SGMII interface capabilities, a DDR2/DDR3 SDRAM memory controller, a 32-bit PCI controller, a programmable interrupt controller, an enhanced serial peripheral interface (eSPI), an enhanced secure digital host controller (eSDHC), three USB dual-role controllers (host/device), two I$^2$C controllers, a four-channel DMA controller, an enhanced local bus controller (eLBC), an integrated security engine with XOR acceleration, a general-purpose I/O port, and dual universal asynchronous receiver/transmitters (DUART). For high speed interconnect, the MPC8536E provides a set of multiplexed pins that support up to three PCI Express interfaces through a dedicated SerDes. The high level of integration in the MPC8536E helps simplify board design and offers significant bandwidth and performance.

The MPC8536E is also available without a security engine, in a configuration known as the MPC8536. All specifications other than those relating to security apply to the MPC8536 exactly as described in this document.

## 1.2 MPC8536E Overview

This section provides a high-level overview of MPC8536E. Figure 1-1 shows the major functional units within the device.

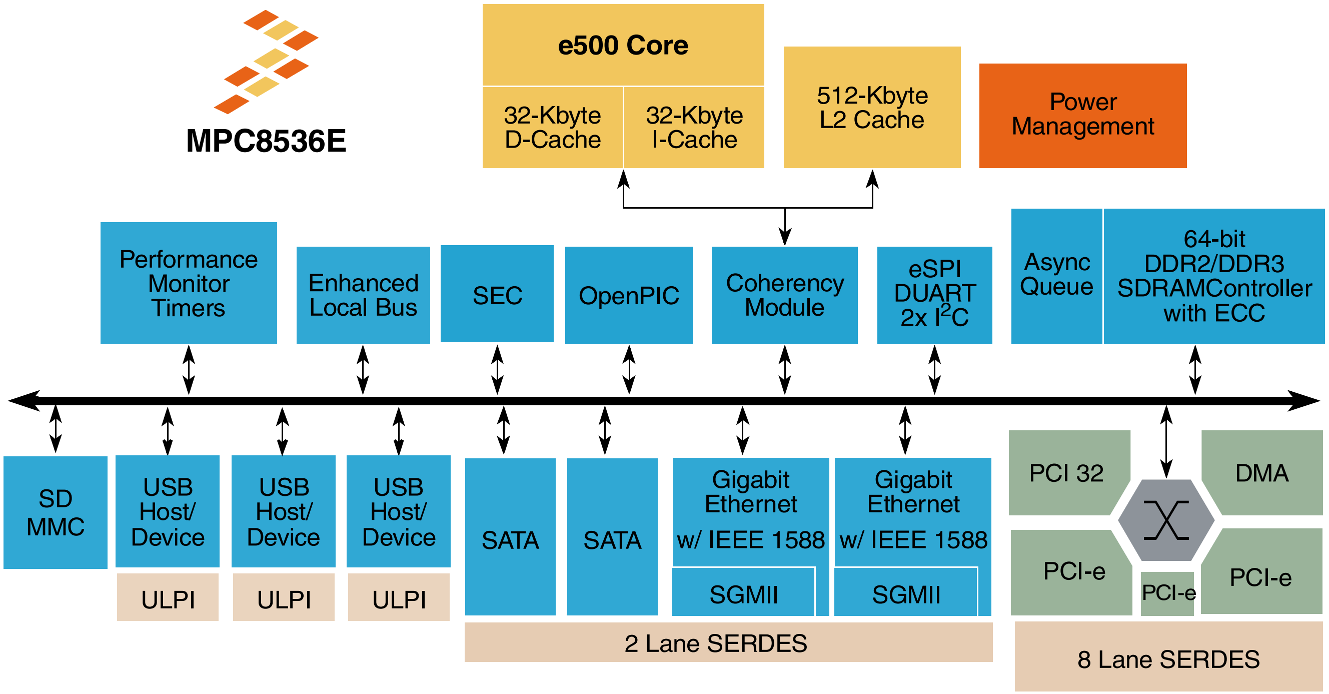


**Figure 1-1. MPC8536E Block Diagram**

### 1.2.1 Key Features

Key features of the MPC8536E include:

- High-performance PowerPC e500v2 core with 36-bit physical addressing
- 512-Kbyte L2 cache with ECC
- Low power consumption
- Integrated security engine (SEC) with XOR acceleration
- Advanced power management controller
- Jog mode feature allows core frequency to be adjusted to optimize power consumption
- Three USB dual-role controllers (host/device)
- Two integrated 10/100/1000 Mb enhanced three-speed Ethernet controllers (eTSECs) with TCP/IP acceleration and classification capabilities. Additional features include
  - — SGMII (serial GMII) interface support, through the SerDes interface)
  - — Hardware support for IEEE Std. 1588™ precision time protocol
- Three PCI Express® controllers utilizing the SerDes interface
- Two serial ATA interfaces
- DDR2/DDR3 SDRAM memory controller

- Programmable interrupt controller (PIC)
- Enhanced serial peripheral interface (eSPI)
- Enhanced secure digital host controller (eSDHC)
- Four-channel DMA controller
- Two I$^2$C controllers
- DUART
- Enhanced local bus controller (eLBC)
- 16 general-purpose I/O signals (independently configurable)
- Package pinout for low-cost PCB

## 1.3 MPC8536E Architecture Overview

The following sections describe the major functional units of the MPC8536E.

### 1.3.1 e500 Core and Memory Unit

The MPC8536E contains a high-performance 32-bit Book E-enhanced e500v2 core that implements the PowerPC architecture.

In addition to 36-bit physical addressing, this version of the e500 core includes:

- Double-precision floating-point APU with an instruction set for double-precision (64-bit) floating-point instructions that use the 64-bit general-purpose registers.
- Embedded vector and scalar single-precision floating-point APUs with an instruction set for single-precision (32-bit) floating-point instructions.

The MPC8536E contains a 512-Kbyte L2 cache/SRAM, as follows:

- Eight-way set-associative cache organization with 32-byte cache lines.
- Per-way allocation of cache region to a given processor.
- Flexible configuration (one, two, four, or eight ways can be configured as SRAM).
- External masters can force data to be allocated into the cache through programmed memory ranges or special transaction types (stashing).
- SRAM:
  - I/O devices access SRAM regions by marking transactions as snoopable (global).
  - Regions can reside at any aligned location in the memory map.
  - Byte-accessible ECC uses read-modify-write accesses for smaller-than-cache-line accesses.

### 1.3.2 e500 Coherency Module (ECM) and Address Map

The e500 coherency module (ECM) provides a mechanism for I/O-initiated transactions to snoop the bus between the e500 core and the integrated L2 cache in order to maintain coherency across local cacheable memory. It also provides a flexible switch-type structure for core- and I/O-initiated transactions to be routed or dispatched to target modules on the device.

The MPC8536E supports a flexible 36-bit physical address map. Conceptually, the address map consists of local space and external address space. The local address map is supported by ten local access windows that define mapping within the local 36-bit (64-Gbyte) address space.

The MPC8536E can be made part of a larger system address space through the mapping of translation windows. This functionality is included in the address translation and mapping units (ATMUs). Both inbound and outbound translation windows are provided. The ATMUs allows the MPC8536E to be part of larger address maps such as the PCI Express 64-bit address environment.

### 1.3.3 Integrated Security Engine (SEC)

The SEC 3.0 off-loads computationally intensive security functions, such as key generation and exchange, authentication, and bulk encryption from the processor cores of the MPC8536E. It is optimized to process all cryptographic algorithms associated with IPsec, IKE, SSL/TLS, iSCSI, SRTP, 802.11i, 3G, A5/3 for GSM and EDGE, and GEA3 for GPRS. The SEC 3.0 derives from integrated security cores in other members of the PowerQUICC family, including the SEC 2.1 Rev2 in the MPC8548E and the SEC 1.0 in the MPC8272E.

Components of the SEC are as follows:

- XOR engine for parity checking in RAID storage applications. Also, the exclusive OR (XOR) operation to generate parity data in RAID applications can be accelerated. XOR operations use SEC descriptors and offload both parity generation and data movement from the e500 core.
- Four crypto-channels, each supporting multi-command descriptor chains.
- Eight cryptographic execution units:
  - Advanced Encryption Standard unit (AESU)
  - ARC four execution unit (AFEU)
  - Cyclic redundancy check accelerator (CRCA)
  - Data Encryption Standard execution unit (DEU)
  - Kasumi execution unit (KEU)
  - Message digest execution unit (MDEU)
  - Public key execution unit (PKEU)
  - Random number generator (RNGB)

### 1.3.4 High-Speed Interface Blocks (SerDes)

The MPC8536E offers two high-speed SerDes interface blocks. These blocks are connected to the SGMII, PCI Express, and SATA interfaces as described in this section.

### 1.3.4.1 Eight-Lane SerDes

The eight-lane SerDes allows use of the three PCI Express controllers. One of the configurations in Table 1-1 can be selected during power-on reset as described in Section 4.4.3.8, "SerDes1 I/O Port Selection."

**Table 1-1. Supported SerDes 1 (PCI Express) Configurations**

| PCI Express Signal/Lane | | | | | | | |
|---|---|---|---|---|---|---|---|
| **0/A** | **1/B** | **2/C** | **3/D** | **4/E** | **5/F** | **6/G** | **7/H** |
| PEX1 x8[1] | | | | | | | |
| PEX1 x4 | | | | PEX2 x4 | | | |
| PEX1 x4 | | | | PEX2 x2 | | PEX3 x2 | |

[1] 8-Bit PCI Express is only available at platform frequency of 527 MHz or greater.

### 1.3.4.2 Two-Lane SerDes

The two-lane SerDes allows use of the SATA controllers or of the SGMII interfaces of the eTSEC controllers (selected at power-on reset). Both lanes must be either SATA or SGMII. See Section 4.4.3.9, "SerDes2 I/O Port Selection," for configuration options.

## 1.3.5 Enhanced Three-Speed Ethernet Controllers (eTSEC)

Two MPC8536E on-chip enhanced three-speed Ethernet controllers (eTSECs) incorporate a media access control (MAC) sublayer that supports 10 and 100 Mbps and 1 Gbps Ethernet/802.3 networks with MII, RMII, GMII, RGMII, TBI, and RTBI physical interfaces as well as SGMII interfaces through a dedicated SerDes. The eTSECs include 2 Kbyte receive and 10 Kbyte transmit FIFOs and DMA functions.

The MPC8536E eTSECs support programmable CRC generation and checking, RMON statistics, and jumbo frames of up to 9.6 Kbytes. Frame headers and buffer descriptors (BDs) can be forced into the L2 cache to speed classification or other frame processing. They are designed to comply with IEEE Std. 802.3®, 802.3u®, 802.3x®, 802.3z®, 802.3ac®, 802.3ab®. The BDs are based on the MPC8260 and MPC860T 10/100 Ethernet programming models. Each eTSEC provides hardware support for accelerating TCP/IP packet transmission and reception. By default, TCP/IP acceleration is not enabled and the eTSEC processes frames as pure Ethernet frames, emulating a PowerQUICC III TSEC and allowing existing driver software to be re-used with minimal change. Key features of these controllers include:

- Flexible configuration for multiple PHY interface configurations. The SGMII interface is available for any combination of eTSECs, regardless of the configuration of any other eTSEC.

  Table 1-2 lists available configurations for eTSEC1 and 3.

**Table 1-2. Supported eTSEC1 and eTSEC3 Configurations**

| Mode Option | eTSEC1 | eTSEC3 |
|---|---|---|
| Ethernet standard interfaces | TBI, GMII, or MII | TBI, GMII, or MII |
| Ethernet reduced interfaces | RTBI, RGMII, RMII, or SGMII | RTBI, RGMII, RMII or SGMII |
| FIFO interface | 8-bit FIFO | 8-bit FIFO |

- TCP/IP acceleration and QoS features:
  - IP v4 and IP v6 header recognition on receive
  - IP v4 header checksum verification and generation
  - TCP and UDP checksum verification and generation
  - Per-packet configurable acceleration
  - Recognition of VLAN, stacked (queue in queue) VLAN, 802.2, PPPoE session, MPLS stacks, and ESP/AH IP-security headers
  - All FIFO modes
  - Transmission from up to eight physical queues
  - Reception to up to eight physical queues
- Full- and half-duplex Ethernet support (1000 Mbps supports only full duplex): IEEE Std. 802.3 full-duplex flow control (automatic PAUSE frame generation or software-programmed PAUSE frame generation and recognition)
- IEEE Std. 802.1® virtual local area network (VLAN) tags and priority
- VLAN insertion and deletion
  - Per-frame VLAN control word or default VLAN for each eTSEC
  - Extracted VLAN control word passed to software separately
- Programmable Ethernet preamble insertion and extraction of up to 7 bytes
- MAC address recognition
- Can force allocation of header information and buffer descriptors into L2 cache
- Supports IEEE 1588 precision time protocol for network synchronization over Ethernet

### 1.3.6 Universal Serial Bus (USB) Dual-Role Controllers

The three USB dual-role controllers comply with USB specification revision 2.0.

#### 1.3.6.1 Host Mode Operation

- Support operation as stand-alone USB host controllers
  - Support USB root hub with one downstream-facing port
  - Enhanced host controller interface (EHCI) compatible
- Support high-speed (480 Mbps), full-speed (12 Mbps), and low-speed (1.5 Mbps) operations
- Support a direct connection to a high-speed device without an external hub
- Support external PHY with UTMI+ low-pin interface (ULPI)

#### 1.3.6.2 Device Mode Operation

- Support operation as a stand-alone USB device
  - Support one upstream facing port
  - Support six programmable USB endpoints

- Support high-speed (480 Mbps) and full-speed (12 Mbps)
- Support external PHY with UTMI+ low-pin interface (ULPI)

### 1.3.7 DDR SDRAM Controller

The DDR memory controller supports DDR2 and DDR3 SDRAM. The memory interface controls main memory accesses and provides a maximum of 32 Gbyte of main memory. The MPC8536E also supports chip-select interleaving and controller interleaving. There is a variety of MPC8536E SDRAM configurations. SDRAM banks can be built using DIMMs or directly-attached memory devices. Sixteen multiplexed address signals provide for device densities of 64 Mbits, 128 Mbits, 256 Mbits, 512 Mbits, 1 Gbit, 2 Gbit, and 4 Gbit. Four chip-select signals support up to four banks of memory. Bank sizes range from 64 Mbyte to 4 Gbyte. Nine column address strobes (D*n*_MDM[0:8]) provide byte selection for memory bank writes. There is cache line and page interleaving between memory controllers.

The MPC8536E can be configured to retain the currently active SDRAM page for pipelined burst accesses. Page mode support of up to 32 simultaneously open pages can dramatically reduce access latencies for page hits. Depending on the memory system design and timing parameters, page mode can save 3 to 4 clock cycles for subsequent burst accesses that hit in an active page.

Using ECC, the MPC8536E detects and corrects all single-bit errors and detects all double-bit errors and all errors within a nibble.

The MPC8536E can invoke a level of system power management by asserting the D*n*_MCKE SDRAM signal on-the-fly to put the memory into a low-power sleep mode.

The DDR controllers offer both hardware and software options for battery-backed main memory. In addition, the DDR controllers offer an initialization bypass feature for use by system designers to prevent re-initialization of main memory during system power-on after an abnormal shutdown.

### 1.3.8 Power Management Controller

The MPC8536E is designed for low power consumption benefitting equipment manufacturers wishing to support ENERGY STAR standards. Dynamic power management locally minimizes power consumption in the doze, nap, or sleep modes. Static power is regulated in the deep sleep mode.

A jog mode feature is provided on the MPC8536E. In jog mode, the e500 core frequency can be adjusted dynamically while the platform frequency remains unchanged, resulting in optimal device temperature and power dissipation.

The power management controller (PMC) is responsible for maintaining the device in various low power modes. The PMC has the following features:

- Low standby power
- Jog mode support—Adjusts core frequency to optimize power consumption
- Fast recovery to restored state
- Support for dynamic and static power management to minimize power consumption of idle blocks:
  - Doze, nap, and sleep modes for dynamic power management
  - Deep sleep mode for static power management

- PMC wake on: LAN activity, USB connection or GPIO, internal timer, or external interrupt event

### 1.3.9 PCI Express Controller

The MPC8536E supports a PCI Express interface compliant with the *PCI Express Base Specification Revision 1.0a*. Each controller is configurable at boot time to act as either root complex or endpoint.

The physical layer of the PCI Express interface operates at a 2.5-Gbaud data rate (effective rate of 2 Gbps due to encoding overhead) per lane. Receive and transmit ports operate independently, resulting in an aggregate theoretical bandwidth of 32 Gbps (x8 link) or 16 Gbps (x4 link).

Other features of the PCI Express interface include:

- x8, x4, x2, and x1 link widths supported. x8 PCI Express is only available at CCB (platform) speed of 527 MHz and above.
- Selectable operation as root complex or endpoint
- Both 32- and 64-bit addressing and 256-byte maximum payload size
- Full 64-bit decode with 36-bit wide windows

### 1.3.10 Programmable Interrupt Controller (PIC)

The MPC8536E PIC implements the logic and programming structures of the OpenPIC architecture, providing for external interrupts (with fully nested interrupt delivery), message interrupts, internal-logic driven interrupts, and global high-resolution timers. Up to 16 programmable interrupt priority levels are supported.

The PIC can be bypassed to allow use of an external interrupt controller.

### 1.3.11 Enhanced Secure Digital Host Controller (eSDHC)

The enhanced secure digital host controller (eSDHC) provides an interface between the host system and SD/MMC cards. The eSDHC acts as a bridge, passing host bus transactions to SD/MMC cards by sending commands and performing data accesses to or from the cards. Under SD protocol, it can be categorized as a memory card, I/O card, or combo card. The memory card invokes a copyright protection mechanism that complies with the security of the SDMI standard.

### 1.3.12 eSPI Interface

The enhanced serial peripheral interface (eSPI) allows the device to operate as an SPI master to exchange data between other PowerQUICC family chips, Ethernet PHYs for configuration, and peripheral devices such as EEPROMs, real-time clocks, A/D converters, and ISDN devices.

The eSPI is a full-duplex, synchronous, character-oriented channel that supports a four-wire interface (receive, transmit, clock, and slave select). The eSPI block consists of transmitter and receiver sections, an independent baud-rate generator, and a control unit. It has the ability to boot from an SPI serial flash device.

The eSPI receiver and transmitter each have a FIFO of 32 bytes to support RapidS™ for Atmel™ devices. The eSPI also supports Winbond™ dual-output read commands; in this mode the eSPI uses two bits in parallel for read.

### 1.3.13 Serial ATA (SATA) Controllers

The SATA controllers have the following features:

- Support host SATA I per spec Rev 1.0a
  - OOB
  - Port multipliers
  - ATAPI 6+
  - Spread spectrum clocking on receive
- Support for SATA II extensions
  - Asynchronous notification
  - Hot Plug including asynchronous signal recovery
  - Link power management
  - Native command queuing
  - Staggered spin-up and port multiplier support
- Support for SATA I and II data rates
  - 1.5 & 3.0 Gbaud
- Standard ATA master-only emulation
- Includes ATA shadow registers
- Implements SATA superset registers
  - SError, SControl, SStatus
- Interrupt driven
- Power management support
- Error handling and diagnostic features
  - Far end/near end loopback
  - Failed CRC error reporting
  - Increased ALIGN insertion rates
  - Scrambling and CONT override

### 1.3.14 DMA Controller, I$^2$C, DUART, eLBC

Two integrated four-channel DMA controllers can transfer data between any I/O or memory ports or between two devices or locations on the same port. The DMA controllers can be used as follows:

- To chain (both extended and direct) through local memory-mapped chain descriptors.
- To handle misaligned transfers, as well as stride transfers and complex transaction chaining.
- To specify local attributes such as snoop and L2 write stashing.

There are two I$^2$C controllers. These synchronous, multimaster buses can be connected to additional devices for expansion and system development.

The DUART supports full-duplex operation and is compatible with the PC16450 and PC16550 programming models. 16-byte FIFOs are supported for both the transmitter and the receiver.

The enhanced local bus controller (eLBC) port connects to a wide variety of external memories, DSPs, and ASICs. Three separate state machines share the same external pins and can be programmed separately to access different types of devices. The general-purpose chip select machine (GPCM) controls accesses to asynchronous devices using a simple handshake protocol. The user-programmable machine (UPM) can be programmed to interface to synchronous devices or custom ASIC interfaces. Features of the local bus controller are as follows:

- Multiplexed 32-bit address and data bus operating at up to 133 MHz
- Eight chip selects for eight external slaves
- Up to eight-beat burst transfers
- 32-, 16-, and 8-bit port sizes controlled by an internal memory controller
- Three protocol engines on a per-chip-select basis
- Parity support
- Default boot ROM chip select with configurable bus width (8, 16, or 32 bits)

## 1.4 MPC8536E Application Examples

### 1.4.1 Multi-Function Printer

Figure 1-2 illustrates how the MPC8536E can be implemented in a high-speed color printer application. In this application, the CPU interfaces to the print and scan ASIC through the PCI Express interfaces. Image data from the scanner, fax, or network is processed by CPU and the math accelerators on the MPC8536E before being sent to the printer engine. High-speed color processing and concurrency of application in MFP systems require the higher processor performance and fast data movement provided by MPC8536E in order to manipulate large, high-quality images at high speeds. MPC8536E implements advanced power management methods to minimize the power consumption.

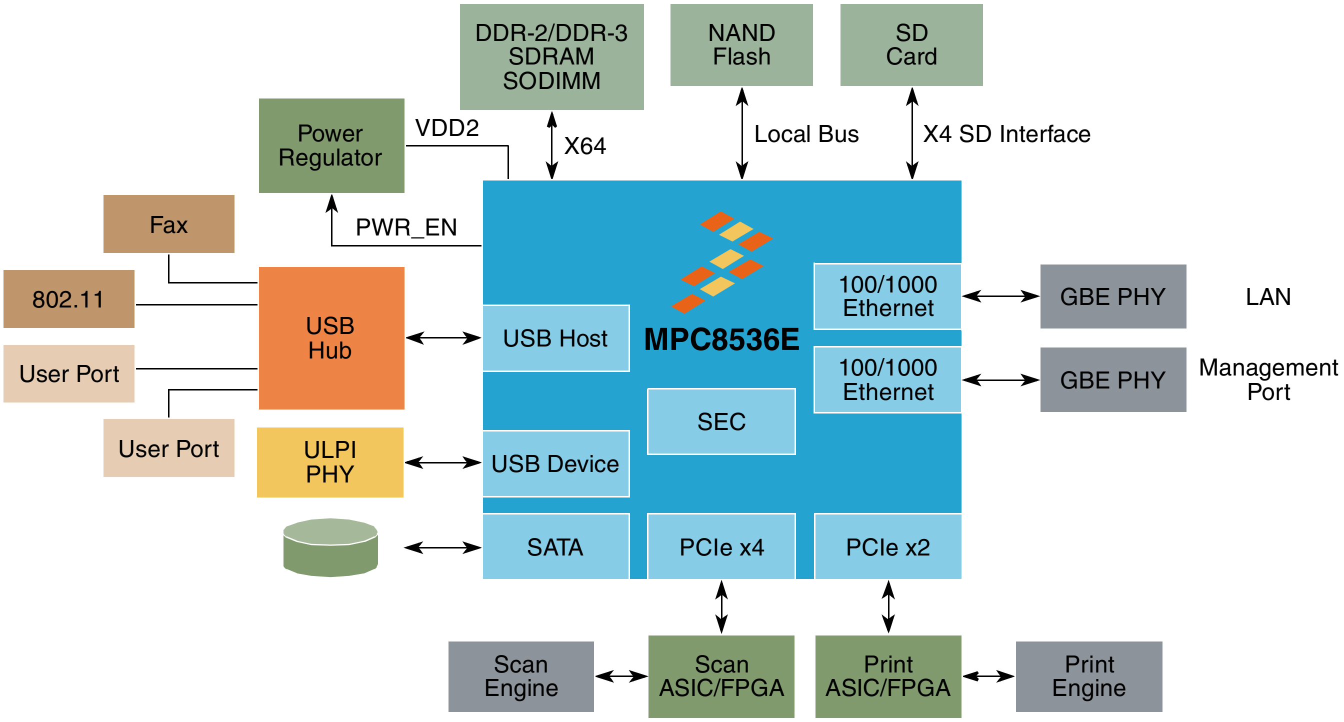


Figure 1-2. Multi-Function Printer Application

### 1.4.2 Network Attached Storage

Figure 1-3 illustrates how a network attached storage application can be realized with the MPC8536E. In this application, the MPC8536E PCI Express can be configured for high-speed, 8 lane configuration, which can support high data rate RAID interfaces. For network connectivity, dual Gigabit Ethernet controllers provide high bandwidth to the storage medium. The security engine provides acceleration for IPSec as well as XOR acceleration for RAID parity calculations, and CRC32C digest calculations for iSCSI applications. The platform also supports battery back-up for data protection in the event of power loss.

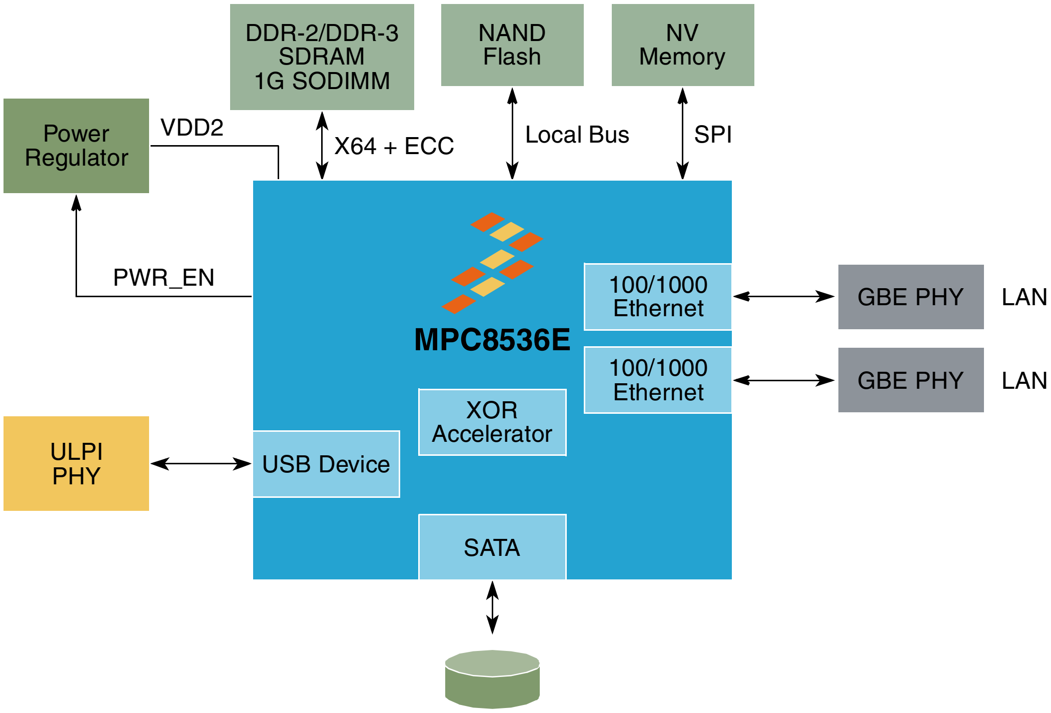


Figure 1-3. Network Attached Storage Application

### 1.4.3 Gaming Kiosk

Figure 1-4 illustrates a highly integrated implementation of the MPC8536E in a gaming and information kiosk application. Gaming systems require fast responsive controls and vibrant graphics processing. MPC8536E PCI Express provides the high bandwidth interface to transfer graphics image data and control to the Graphics Processor. Gaming systems are mechanically secure and require low power operation. MPC8536E GHz performance in low power envelopes provides the mechanism for the designer to create high performance in a fanless system.

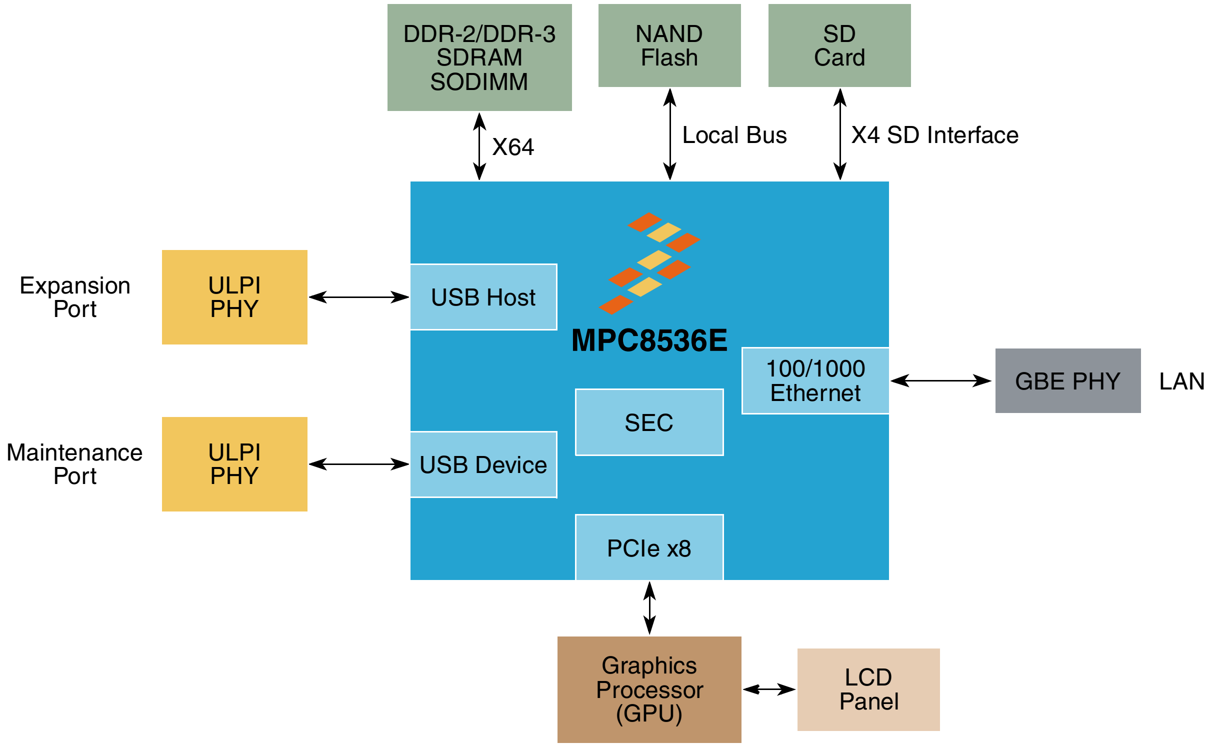


Figure 1-4. Gaming Kiosk Application

### 1.4.4 Network Controller

Figure 1-5 illustrates how a network controller application can be realized with the MPC8536E. One of the gigabit Ethernet controllers can be used to interface to an Ethernet switch. The Ethernet interfaces allow the option for SGMII, which provide robust low-power connectivity to PHY devices. MPC8536E also implements IEEE 1588 for network synchronization.

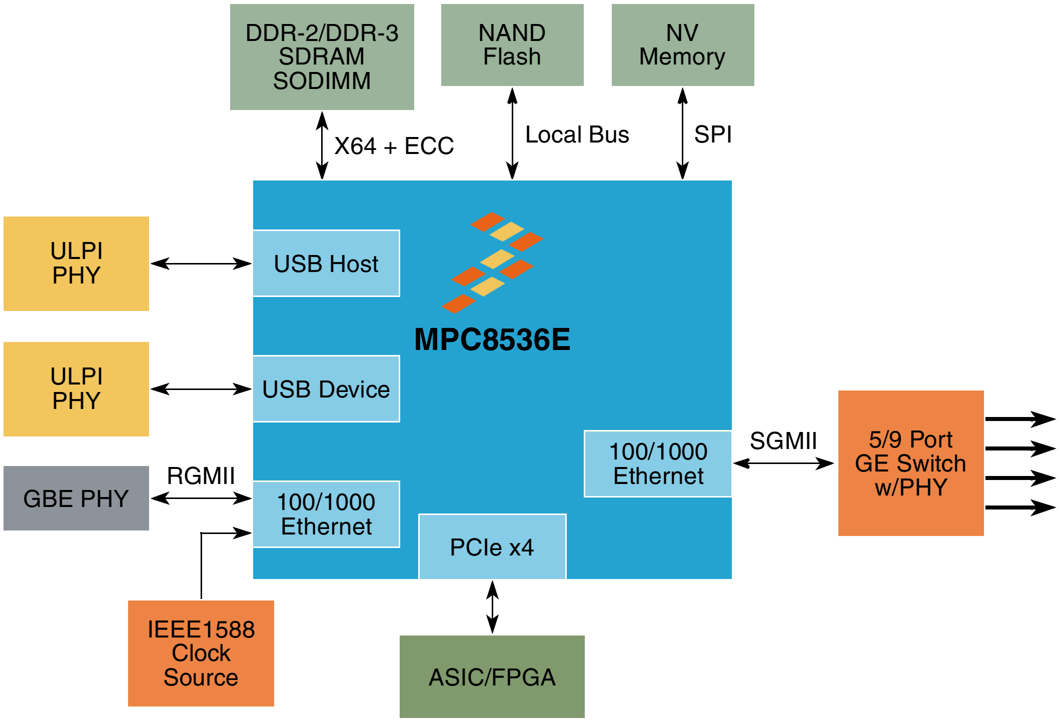


Figure 1-5. Network Controller Application

# Chapter 2
# Memory Map

This chapter describes the MPC8536 memory map. An overview of the local address map is followed by a description of how local access windows are used to define the local address map. The inbound and outbound address translation mechanisms used to map to and from external memory spaces are described next. Finally, the configuration, control, and status registers are described, including a complete listing of all memory-mapped registers with cross references to the sections detailing descriptions of each.

## 2.1 Local Memory Map Overview and Example

The MPC8536 provides an extremely flexible local memory map. The local memory map refers to the 36-bit address space seen by the processor as it accesses memory and I/O space. DMA engines also see this same local memory map. All memory accessed by the DDR SDRAM and local bus memory controllers exists in this memory map, as do all memory-mapped configuration, control, and status registers.

The local memory map is defined by a set of 12 local access windows. Each of these windows maps a region of memory to a particular target interface, such as the DDR SDRAM controller or the PCI controller. Note that the local access windows do not perform any address translation. The size of each window can be configured from 4 Kbytes to 32 Gbytes. The target interface is specified using the codes shown in Table 2-1.

**Table 2-1. Target Interface Codes**

| Source/Target Interface | Target Code |
|---|---|
| PCI | 00000 |
| PCI Express 2 | 00001 |
| PCI Express 1 | 00010 |
| PCI Express 3 | 00011 |
| Enhanced local bus | 00100 |
| Configuration space | 01000 |
| DDR SDRAM | 01111 |

Figure 2-1 shows an example memory map.

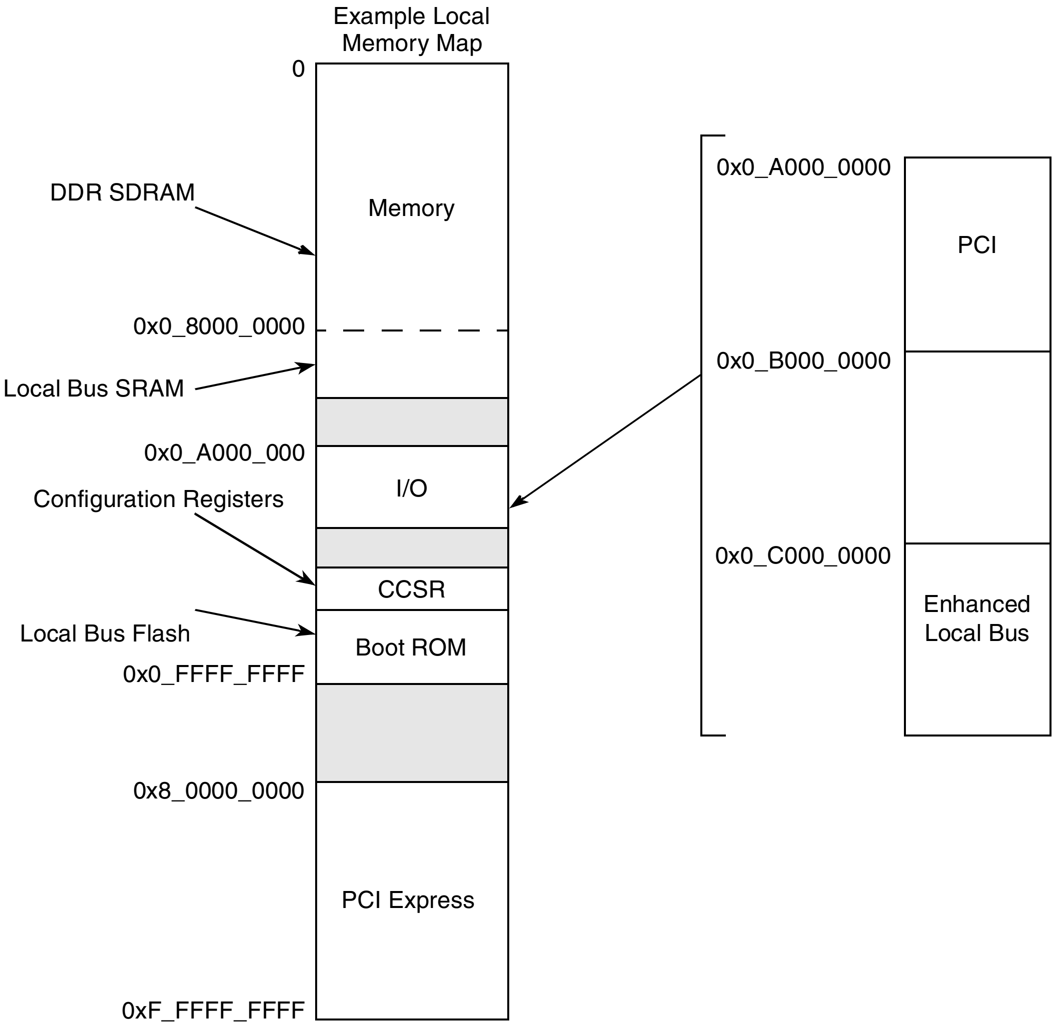


**Figure 2-1. Local Memory Map Example**

Table 2-2 shows one corresponding set of local access window settings.

**Table 2-2. Local Access Windows Example**

| Window | Base Address | Size | Target Interface |
|---|---|---|---|
| 0 | 0x0_0000_0000 | 2 Gbytes | 0b1111 (DDR SDRAM) |
| 1 | 0x0_8000_0000 | 1 Mbyte | 0b0100 (Enhanced local bus) |
| 2 | 0x0_A000_0000 | 256 Mbytes | 0b0000 (PCI) |
| 3 | 0x0_C000_0000 | 256 Mbytes | 0b0100 (Enhanced local bus) |
| 4 | 0x8_0000_0000 | 32 Gbytes | 0b0100 (PCI Express) |
| 5–9 | Unused | | |

In this example, it is not necessary to use a local access window to specify the range of memory used for memory-mapped registers because this is a fixed 1-Mbyte space pointed to by CCSRBAR. See Section 4.3.1.1.2, “Configuration, Control, and Status Registers Base Address Register (CCSRBAR).” Neither is it required to define a local access window to describe the location of the boot ROM because it is in the default location (see Section 4.4.3.6, “Boot ROM Location”). However, note that the e500 core only provides one default TLB entry to access boot code and it allows for accesses within the highest 4 Kbytes of the low 4 Gbytes of memory. In order for the e500 to access the full 8 Mbytes of default boot space (and the 1 Mbyte of CCSR space), additional TLB entries must be set up within the e500 MMU for mapping these regions.

## 2.2 Address Translation and Mapping

Four distinct types of translation and mapping operations are performed on transactions in the MPC8536. These are as follows:

- Mapping a local address to a target interface
- Assigning attributes to transactions
- Translating the local 36-bit address to an external address space
- Translating external addresses to the local 36-bit address space

The local access windows perform target mapping for transactions within the local address space. No address translation is performed by the local access windows.

Outbound ATMU windows perform the mapping from the local 36-bit address space to the address spaces of PCI or PCI-Express, for example, which may be much larger than the local space. Outbound ATMU windows also map attributes such as transaction type or priority level.

Inbound ATMU windows perform the address translation from the external address space to the local address space, attach attributes and transaction types to the transaction, and also map the transaction to its target interface. Note that in mapping the transaction to the target interface, an inbound ATMU window performs a similar function as the local access windows. The target mappings created by an inbound ATMU must be consistent with those of the local access windows. That is, if an inbound ATMU maps a transaction to a given local address and a given target, a local access window must also map that same local address to the same target.

All of the configuration registers that define translation and mapping functions use the concept of translation or mapping windows, and all follow the same register format. Table 2-3 summarizes the general format of these window definitions.

**Table 2-3. Format of ATMU Window Definitions**

| Register | Function |
|---|---|
| Translation address | High-order address bits defining location of the window in the target address space |
| Base address | High-order address bits defining location of the window in the initial address space |
| Window size/attributes | Window enable, window size, target interface, and transaction attributes |

Windows must be a power-of-two size. To perform a translation or mapping function, the address of the transaction is compared with the base address register of each window. The number of bits used in the comparison is dictated by each window's size attribute. When an address hits a window, if address translation is being performed, the new translated address is created by concatenating the window offset to the translation address. Again, the window's size attribute dictates how many bits are translated.

### 2.2.1 SRAM Windows

The on-chip memory array of the MPC8536 can be configured as a memory-mapped SRAM ranging from 64 Kbytes to 256 Kbytes. Configuration registers in the L2 cache controller set the base addresses and sizes for these windows. When enabled, these windows supersede all other mappings of these addresses for processor and global (snoopable) I/O transactions. Therefore, SRAM windows must never overlap configuration space as defined by CCSRBAR. It is possible to have SRAM windows overlap local access windows, but this is discouraged because processor and snoopable I/O transactions would map to the SRAM while non-snooped I/O transactions would be mapped by the local access windows. Only if all accesses to the SRAM address range are snoopable can results be consistent if the SRAM window overlaps a local access window.

See Section 6.3.1.3.1, "L2 Memory-Mapped SRAM Base Address Registers 0–1 (L2SRBARn)," for information about configuring SRAM windows.

### 2.2.2 Window into Configuration Space

CCSRBAR defines a window used to access all memory-mapped configuration, control, and status registers. No address translation is done, so there are no associated translation address registers. The window is always enabled with a fixed size of 1 Mbyte; no other attributes are attached, so there is no associated size/attribute register. This window always takes precedence over all local access windows. See Section 4.3.1.1.2, "Configuration, Control, and Status Registers Base Address Register (CCSRBAR)," and Section 2.3, "Configuration, Control, and Status Register Map."

### 2.2.3 Local Access Windows

As demonstrated in the address map overview in Section 2.1, "Local Memory Map Overview and Example," local access windows associate a range of the local 36-bit address space with a particular target interface. This allows the internal interconnections of the MPC8536 to route a transaction from its source to the proper target. No address translation is performed. The base address defines the high order address bits that give the location of the window in the local address space. The window attributes enable the window, define its size, and specify the target interface.

With the exception of configuration space (mapped by CCSRBAR), on-chip SRAM regions (mapped by L2SRBAR registers), and default boot ROM, all addresses used by the system must be mapped by a local access window. This includes addresses that are mapped by inbound ATMU windows; target mappings of inbound ATMU windows and local access windows must be consistent.

The local access window registers exist as part of the local access block in the general utilities registers. See Section 2.3.4, "General Utilities Registers." A detailed description of the local access window registers is given in the following sections. Note that the minimum size of a window is 4 Kbytes, so the low order 12 bits of the base address cannot be specified.

### 2.2.3.1 Local Access Register Memory Map

Table 2-4 shows the memory map for the local access registers. In this table and in the register figures and field descriptions, the following access definitions apply:

- Reserved fields are always ignored for the purposes of determining access type.
- R/W, R, and W (read/write, read only, and write only) indicate that all the non-reserved fields in a register have the same access type.
- w1c indicates that all of the non-reserved fields in a register are cleared by writing ones to them.
- Mixed indicates a combination of access types.
- Special is used when no other category applies. In this case the register figure and field description table should be read carefully.

**Table 2-4. Local Access Register Memory Map**

| Local Memory Offset (Hex) | Register | Access | Reset | Section/Page |
|---|---|---|---|---|
| 0x0_0BF8 | LAIPBRR1—Local access IP block revision register 1 | R | 0x0000_0000 | 2.2.3.2/2-6 |
| 0x0_0BFC | LAIPBRR2—Local access IP block revision register 2 | R | 0x0000_0000 | 2.2.3.3/2-6 |
| 0x0_0C08 | LAWBAR0—Local access window 0 base address register | R/W | 0x0000_0000 | 2.2.3.4/2-7 |
| 0x0_0C10 | LAWAR0—Local access window 0 attribute register | R/W | 0x0000_0000 | 2.2.3.5/2-7 |
| 0x0_0C28 | LAWBAR1—Local access window 1 base address register | R/W | 0x0000_0000 | 2.2.3.4/2-7 |
| 0x0_0C30 | LAWAR1—Local access window 1 attribute register | R/W | 0x0000_0000 | 2.2.3.5/2-7 |
| 0x0_0C48 | LAWBAR2—Local access window 2 base address register | R/W | 0x0000_0000 | 2.2.3.4/2-7 |
| 0x0_0C50 | LAWAR2—Local access window 2 attribute register | R/W | 0x0000_0000 | 2.2.3.5/2-7 |
| 0x0_0C68 | LAWBAR3—Local access window 3 base address register | R/W | 0x0000_0000 | 2.2.3.4/2-7 |
| 0x0_0C70 | LAWAR3—Local access window 3 attribute register | R/W | 0x0000_0000 | 2.2.3.5/2-7 |
| 0x0_0C88 | LAWBAR4—Local access window 4 base address register | R/W | 0x0000_0000 | 2.2.3.4/2-7 |
| 0x0_0C90 | LAWAR4—Local access window 4 attribute register | R/W | 0x0000_0000 | 2.2.3.5/2-7 |
| 0x0_0CA8 | LAWBAR5—Local access window 5 base address register | R/W | 0x0000_0000 | 2.2.3.4/2-7 |
| 0x0_0CB0 | LAWAR5—Local access window 5 attribute register | R/W | 0x0000_0000 | 2.2.3.5/2-7 |
| 0x0_0CC8 | LAWBAR6—Local access window 6 base address register | R/W | 0x0000_0000 | 2.2.3.4/2-7 |
| 0x0_0CD0 | LAWAR6—Local access window 6 attribute register | R/W | 0x0000_0000 | 2.2.3.5/2-7 |
| 0x0_0CE8 | LAWBAR7—Local access window 7 base address register | R/W | 0x0000_0000 | 2.2.3.4/2-7 |
| 0x0_0CF0 | LAWAR7—Local access window 7 attribute register | R/W | 0x0000_0000 | 2.2.3.5/2-7 |
| 0x0_0D08 | LAWBAR8—Local access window 8 base address register | R/W | 0x0000_0000 | 2.2.3.4/2-7 |
| 0x0_0D10 | LAWAR8—Local access window 8 attribute register | R/W | 0x0000_0000 | 2.2.3.5/2-7 |
| 0x0_0D28 | LAWBAR9—Local access window 9 base address register | R/W | 0x0000_0000 | 2.2.3.4/2-7 |
| 0x0_0D30 | LAWAR9—Local access window 9 attribute register | R/W | 0x0000_0000 | 2.2.3.5/2-7 |

Table 2-4. Local Access Register Memory Map (continued)

| Local Memory Offset (Hex) | Register | Access | Reset | Section/Page |
|---|---|---|---|---|
| 0x0_0D48 | LAWBAR10—Local access window 10 base address register | R/W | 0x0000_0000 | 2.2.3.5/2-7 |
| 0x0_0D50 | LAWAR10—Local access window 10 attribute register | R/W | 0x0000_0000 | 2.2.3.5/2-7 |
| 0x0_0D68 | LAWBAR11—Local access window 11 base address register | R/W | 0x0000_0000 | 2.2.3.5/2-7 |
| 0x0_0D70 | LAWAR11—Local access window 11 attribute register | R/W | 0x0000_0000 | 2.2.3.5/2-7 |

### 2.2.3.2 Local Access IP Block Revision Register 1 (LAIPBRR1)

The local access IP block revision register 1 is shown in Figure 2-2.

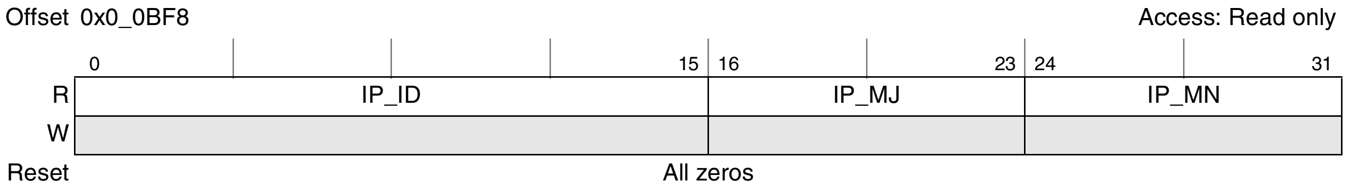


Figure 2-2. Local Access IP Block Revision Register 1 (LAIPBRR1)

Table 2-5 describes LAIPBRR1 fields.

Table 2-5. LAIPBRR1 Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–15 | IP_ID | IP block ID |
| 16–23 | IP_MJ | Major revision |
| 24–31 | IP_MN | Minor revision |

### 2.2.3.3 Local Access IP Block Revision Register 2 (LAIPBRR2)

The local access IP block revision register 2 is shown in Figure 2-3.

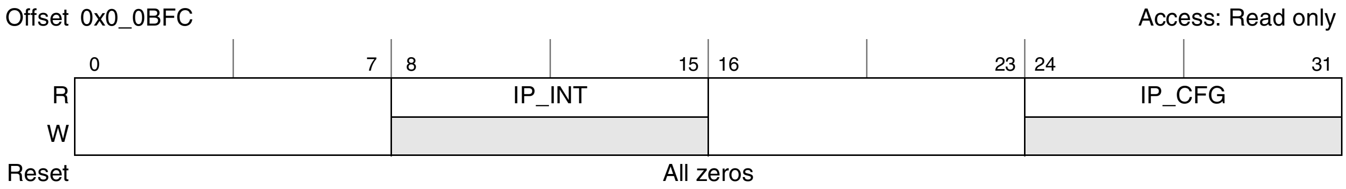


Figure 2-3. Local Access IP Block Revision Register 2 (LAIPBRR2)

Table 2-6 describes LAIPBRR2 fields.

Table 2-6. LAIPBRR2 Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–7 | — | Reserved |
| 8–15 | IP_INT | IP block integration options |

Table 2-6. LAIPBRR2 Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 16–23 | — | Reserved |
| 24–31 | IP_CFG | IP block configuration options |

### 2.2.3.4 Local Access Window *n* Base Address Registers (LAWBAR0–LAWBAR9)

Figure 2-4 shows the bit fields of the LAWBAR*n* registers.

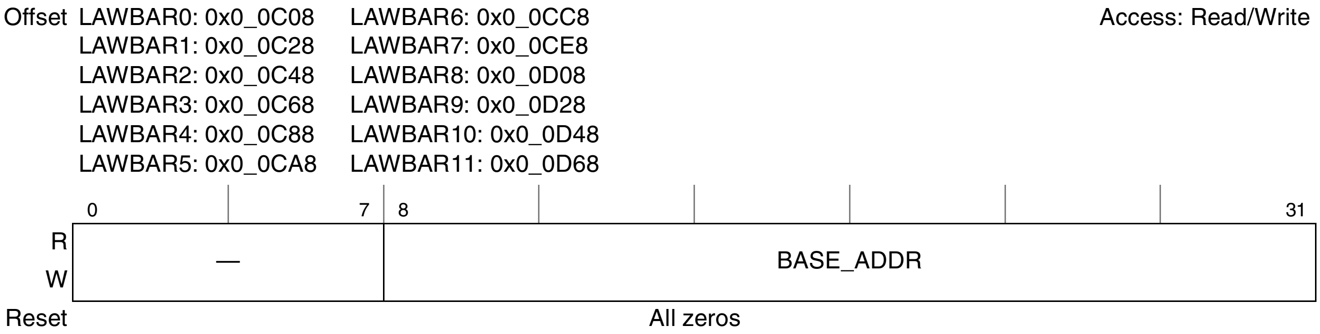


Figure 2-4. Local Access Window *n* Base Address Registers (LAWBAR0–LAWBAR7)

Table 2-7 describes LAWBAR*n* fields.

Table 2-7. LAWBAR*n* Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–7 | — | Write reserved, read = 0 |
| 8–31 | BASE_ADDR | Identifies the 24 most-significant address bits of the base of local access window *n*. The specified base address should be aligned to the window size, as defined by LAWAR*n*[SIZE]. |

### 2.2.3.5 Local Access Window *n* Attributes Registers (LAWAR0–LAWAR9)

Figure 2-5 shows the fields of the LAWAR*n* registers.

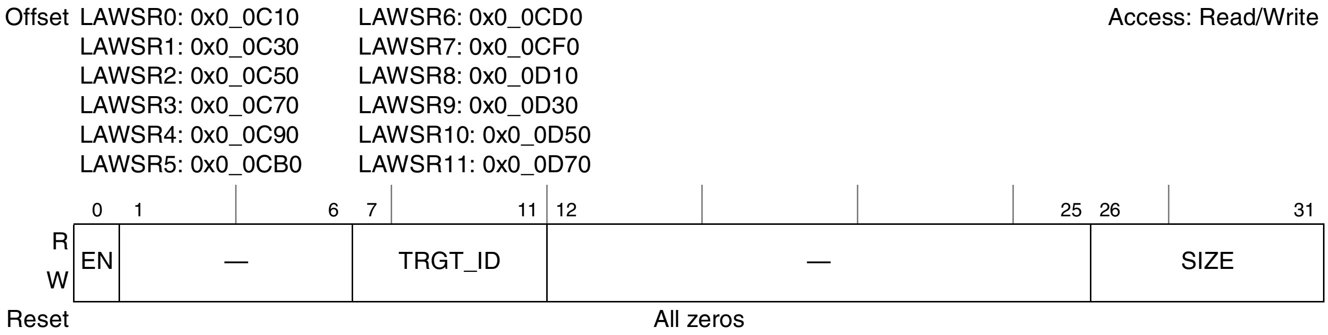


Figure 2-5. Local Access Window *n* Attributes Registers (LAWAR0–LAWAR7)

Table 2-8 describes LAWAR*n* fields.

**Table 2-8. LAWAR*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | EN | 0 The local access window *n* (and all other LAWAR*n* and LAWBAR*n* fields) are disabled.<br>1 The local access window *n* is enabled and other LAWAR*n* and LAWBAR*n* fields combine to identify an address range for this window. |
| 1–6 | — | Write reserved, read = 0 |
| 7–11 | TRGT_ID | Identifies the target interface ID when a transaction hits in the address range defined by this window. Note that configuration registers and SRAM regions are mapped by the windows defined by CCSRBAR and L2SRBAR. These mappings supersede local access window mappings, so configuration registers and SRAM do not appear as a target for local access windows.<br><br>00000 PCI<br>00001 PCI Express 2<br>00010 PCI Express 1<br>00011 PCI Express 3<br>00100 Enhanced local bus<br>0101–1110 Reserved<br>01111 DDR SDRAM |
| 12–25 | — | Write reserved, read = 0 |
| 26–31 | SIZE | Identifies the size of the window from the starting address. Window size is $2^{(SIZE+1)}$ bytes.<br>000000–001010 Reserved<br>001011 4 Kbytes<br>001100 8 Kbytes<br>001101 16 Kbytes<br>. . . . . . $2^{(SIZE+1)}$ bytes<br>100010 32 Gbytes<br>100011–111111 Reserved |

## 2.2.3.6 Precedence of Local Access Windows

If two local access windows overlap, the lower numbered window takes precedence. For instance, if two windows are set up as shown in Table 2-9, local access window 1 governs the mapping of the 1-Mbyte region from 0x0_7FF0_0000 to 0x0_7FFF_FFF, even though the window described in local access window 2 also encompasses that memory region.

**Table 2-9. Overlapping Local Access Windows**

| Window | Base Address | Size | Target Interface |
|---|---|---|---|
| 1 | 0x0_7FF0_0000 | 1 Mbyte | 0b0100 (Local bus controller —LBC) |
| 2 | 0x0_0000_0000 | 2 Gbytes | 0b1111 (DDR SDRAM) |

## 2.2.3.7 Configuring Local Access Windows

Once a local access window is enabled, it should not be modified while any device in the system may be using the window. Neither should a new window be used until the effect of the write to the window is visible to all blocks that use the window. This can be guaranteed by completing a read of the last local access window configuration register before enabling any other devices to use the window. For instance, if local access windows 0–3 are being configured in order during the initialization process, the last write (to LAWAR3) should be followed by a read of LAWAR3 before any devices try to use any of these windows. If the configuration is being done by the local e500 processor, the read of LAWAR3 should be followed by an **isync** instruction.

#### 2.2.3.8 Distinguishing Local Access Windows from Other Mapping Functions

It is important to distinguish between the mapping function performed by the local access windows and the additional mapping functions that happen at the target interface. The local access windows define how a transaction is routed through the MPC8536 internal interconnects from the transactions source to its target. After the transaction has arrived at its target interface, that interface controller may perform additional mapping. For instance, the DDR SDRAM controller has chip select registers that map a memory request to a particular external device. Similarly, the local bus controller has base registers that perform a similar function. The PCI and PCI Express interfaces have outbound address translation and mapping units that map the local address into an external address space.

These other mapping functions are configured by programming the configuration, control, and status registers of the individual interfaces. Note that there is no need to have a one-to-one correspondence between local access windows and chip select regions or outbound ATMU windows. A single local access window can be further decoded to any number of chip selects or to any number or outbound ATMU windows at the target interface.

#### 2.2.3.9 Illegal Interaction Between Local Access Windows and DDR SDRAM Chip Selects

If a local access window maps an address to an interface other than the DDR SDRAM controller, then there should not be a valid chip select configured for the same address in the DDR SDRAM controller. Because DDR SDRAM chip selects boundaries are defined by a beginning and ending address, it is easy to define them so that they do not overlap with local access windows that map to other interfaces.

### 2.2.4 Outbound Address Translation and Mapping Windows

Outbound address translation and mapping refers to the translation of addresses from the local 36-bit address space to the external address space and attributes of a particular I/O interface. On the MPC8536, the following blocks have outbound address translation and mapping units (ATMUs):

- PCI
- PCI Express

The PCI controller has four outbound ATMU windows plus a default window. The PCI outbound ATMU registers include extended translation address registers so that up to 64 bits of external address space can be supported. See Section 16.3.1.2, “PCI ATMU Outbound Registers,” for a detailed description of the PCI outbound ATMU windows.

The PCI Express interface has four outbound ATMU windows plus a default window. The PCI Express outbound ATMU registers include an extended translation address register so that up to 64 bits of external address space can be supported.See Section 17.3.5.1, “PCI Express Outbound ATMU Registers” for a detailed description of the PCI Express outbound ATMU windows.

## 2.2.5 Inbound Address Translation and Mapping Windows

Inbound address translation and mapping refers to the translation of an address from the external address space of an I/O interface (such as PCI address space) to the local 36-bit address space understood by the internal interfaces of the MPC8536. It also refers to the mapping of transactions to a particular target interface and the assignment of transaction attributes. The PCI and the PCI Express controllers have inbound address translation and mapping units (ATMUs).

### 2.2.5.1 PCI Inbound ATMU

The PCI controller has three general inbound ATMU windows plus a dedicated window for memory mapped configuration accesses (PCSRBAR). These windows have a one-to-one correspondence with the base address registers in the PCI programming model. Updating one automatically updates the other. There is no default inbound window; if a PCI address does not match one of the inbound ATMU windows, the device does not respond with an assertion of $\overline{\text{PCI\_DEVSEL}}$. See Section 16.3.1.3, “PCI ATMU Inbound Registers,” for a detailed description of the PCI inbound ATMU windows.

### 2.2.5.2 PCI Express Inbound ATMU

The PCI Express controller has three inbound ATMU windows plus a default. See Section 17.3.5.1, “PCI Express Outbound ATMU Registers,” for a description of the PCI Express inbound ATMU windows.

### 2.2.5.3 Illegal Interaction Between Inbound ATMUs and Local Access Windows

Since both local access windows and inbound ATMUs map transactions to a target interface, it is essential that they not contradict one another. For instance, it is a programming error to have an inbound ATMU map a transaction to the DDR SDRAM memory controller (target interface 0b1111) if the resulting translated local address is mapped to PCI (target interface 0b0000) by a local access window. Such a programming error may result in unpredictable system deadlocks.

## 2.3 Configuration, Control, and Status Register Map

All of the memory mapped configuration, control, and status registers in the MPC8536 are contained within a 1-Mbyte address region. To allow for flexibility, the configuration, control, and status block is relocatable in the local address space. The local address map location of this register block is controlled by the configuration, control, and status registers base address register (CCSRBAR), see Section 4.3.1.1.2, “Configuration, Control, and Status Registers Base Address Register (CCSRBAR).” The default value for CCSRBAR is 4 Gbytes–9 Mbytes, or 0x0_FF70_0000.

**NOTE**

The configuration, control, and status window must not overlap a local access window that maps to the DDR controller. Otherwise, undefined behavior occurs.

### 2.3.1 Accessing CCSR Memory from the Local Processor

When the local e500 processor is used to configure CCSR space, the CCSR memory space should typically be marked as cache-inhibited and guarded.

In addition, many configuration registers affect accesses to other memory regions; therefore writes to these registers must be guaranteed to have taken effect before accesses are made to associated memory regions.

To guarantee that the results of any sequence of writes to configuration registers are in effect, the final configuration register write should be chased by a read of the same register, and that should be followed by a SYNC instruction. Then accesses can safely be made to memory regions affected by the configuration register write.

### 2.3.2 Accessing CCSR Memory from External Masters

In addition to being accessible by the e500 processor, the configuration, control, and status registers are accessible from external interfaces, allowing external masters on the I/O ports to configure the MPC8536.

External masters do not need to know the location of the CCSR memory in the local address map. Rather, they access this region of the local memory map through a window defined by a register in the interface programming model that is accessible to the external master from its external memory map.

The PCI base address for accessing the local CCSR memory is selectable through the PCI configuration and status register base address register (PCSRBAR), at offset 0x10, described in Section 16.3.2.11, “PCI Base Address Registers.” An external PCI master sets this register by running a PCI configuration cycle to the MPC8536. Subsequent memory accesses by a PCI master to the PCI address range indicated by PCSRBAR are translated to the local address indicated by the current setting of CCSRBAR.

### 2.3.3 Organization of CCSR Memory

The configuration, control, and status registers are grouped according to functional units. Most functional blocks are allocated a 4-Kbyte address space for registers. Registers that fall into this category are referred to as general utilities registers. These registers occupy the first 256 Kbytes of CCSR memory.

Registers controlling functions that are not particular to a functional unit but to the device as a whole occupy the highest 256 Kbytes of CCSR memory and are referred to as device-specific registers.

Some functional units such as the OpenPIC-based interrupt controller have larger address spaces as defined by their programming models. The registers for these blocks are given their own large regions of CCSR memory.

**Table 2-10. Local Memory Configuration, Control, and Status Register Summary**

| Offset from CCSRBAR | Register Grouping |
|---|---|
| 0x0_0000–0x3_FFFF | General utilities |
| 0x4_0000–0x7_FFFF | Programmable interrupt controller (PIC) |
| 0x8_0000–0xB_FFFF | Reserved |
| 0xC_0000–0xD_FFFF | Reserved |
| 0xE_0000–0xF_FFFF | Device-specific utilities |

### 2.3.4 General Utilities Registers

Figure 2-6 provides an overview of the general utilities registers.

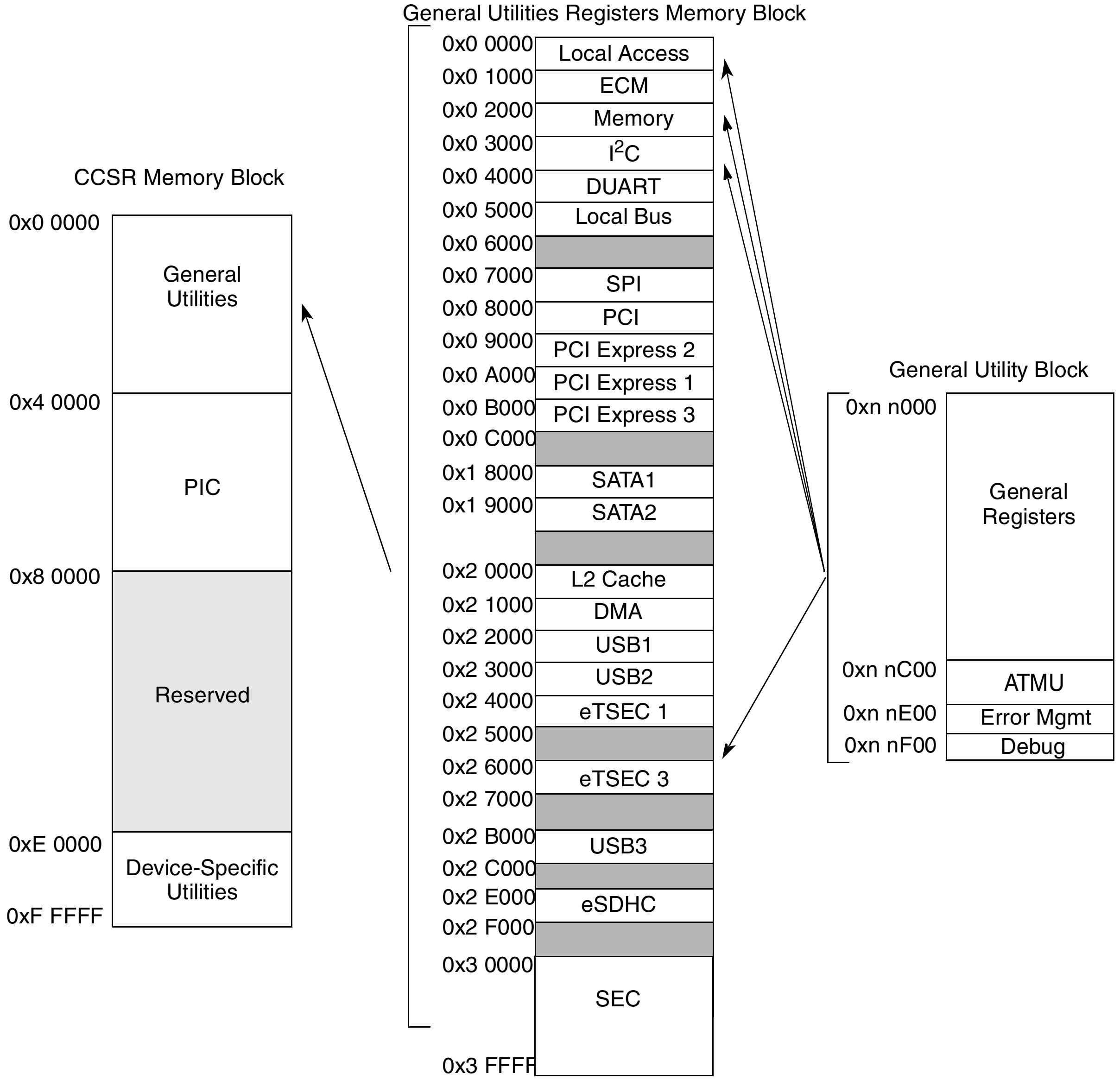


**Figure 2-6. General Utilities Registers Mapping to Configuration, Control, and Status Memory Block**

Figure 2-6 also shows the organization of registers inside the 4-Kbyte register space allocated to an individual functional block. The first 3 Kbytes are available for general registers. The next 512 bytes are dedicated to address translation and mapping registers, if applicable to that particular functional unit (for example, PCI). If a unit has error management registers, they are typically placed starting at offset 0xE00 from the beginning of the block’s 4-Kbyte space, and any debug registers are typically placed in the final 256 bytes of the unit’s register space starting at offset 0xF00.

General utilities registers are accessed as 32-bit quantities except for the DUART and $I^2C$ registers, which are accessed as bytes.

**NOTE**

> Refer to detailed register descriptions for each functional unit for exact locations, sizes, and access requirements. Some blocks may have exceptions to the above guidelines.

## 2.3.5 Interrupt Controller and CCSR

The programmable interrupt controller (PIC) registers are at offset 0x4_0000 from CCSRBAR, see Figure 2-7. Its programming model follows the OpenPIC architecture. The interrupt controller registers should only be accessed with 32-bit accesses.

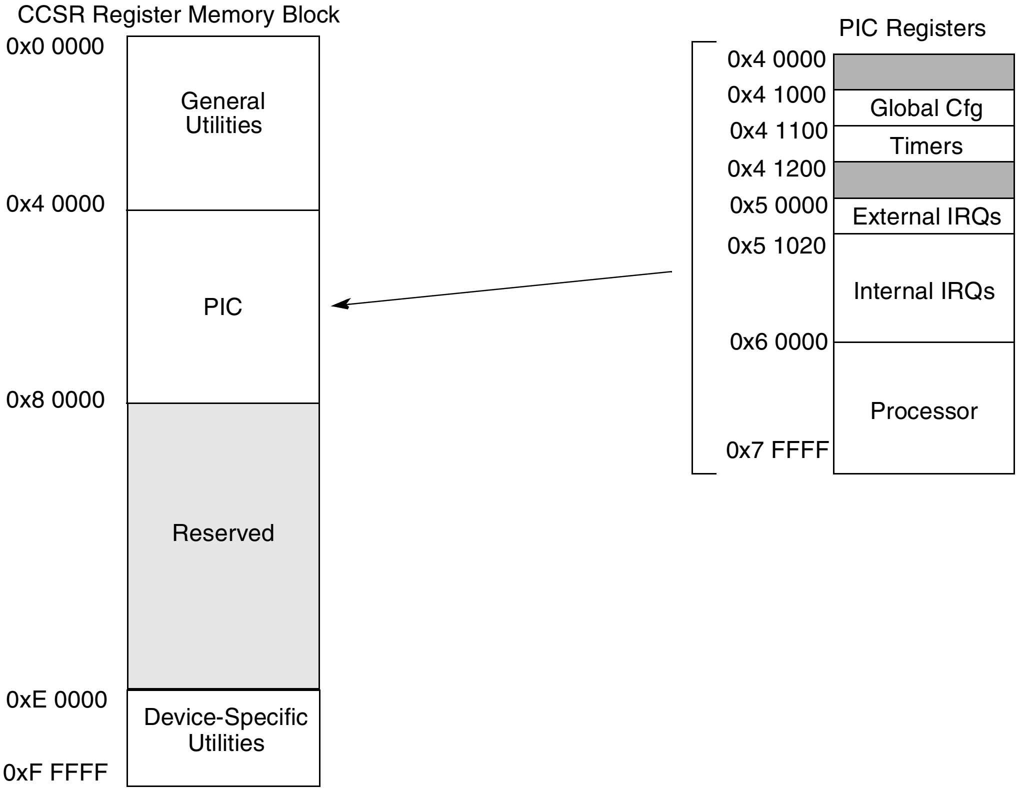


**Figure 2-7. PIC Mapping to Configuration, Control, and Status Memory Block**

## 2.3.6 Device-Specific Utilities

The device-specific registers consist of power management, performance monitors, and device-wide debug utilities (refer to Figure 2-8). These registers are accessible with 32-bit accesses only. Transactions of other than 32-bit are considered a programming error and operation is undefined.

Reserved bits in the following register descriptions are not guaranteed to have predictable values. Software must preserve the values of reserved bits when writing to a register. Also, when reading from a register, software should not rely on the value of any reserved bit remaining consistent.

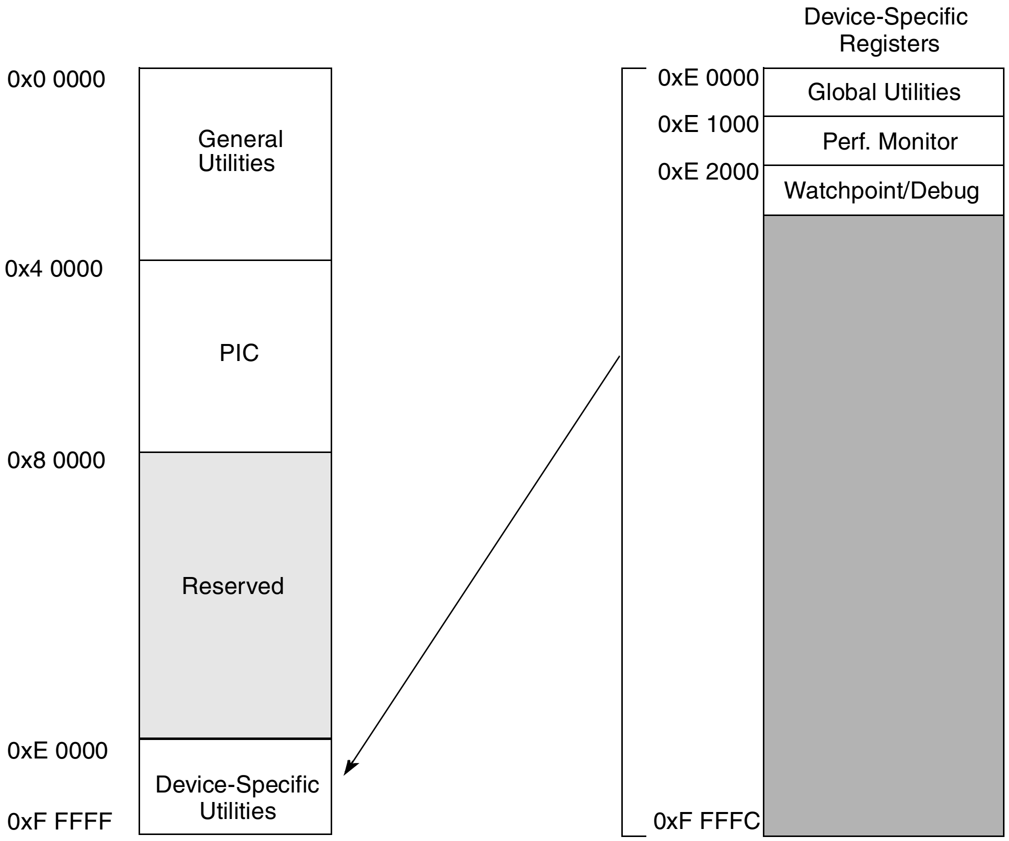


**Figure 2-8. Device-Specific Register Mapping to Configuration, Control, and Status Memory Block**

## 2.4 Complete CCSR Map

Table 2-11 lists the MPC8536 memory-mapped registers.

In this table and in the register figures and field descriptions, the following access definitions apply:

- Reserved fields are always ignored for the purposes of determining access type.
- R/W, R, and W (read/write, read only, and write only) indicate that all the non-reserved fields in a register have the same access type.
- w1c indicates that all of the non-reserved fields in a register are cleared by writing ones to them.
- Mixed indicates a combination of access types.
- Special is used when no other category applies. In this case the register figure and field description table should be read carefully.

Table 2-11. CCSR Block Base Address Map

| Block Base Address (Hex) | Block | Section/Page | Comments |
|---|---|---|---|
| General Utilities (0x0_0000–0x3_FFFF) | | | |
| 0x0_0000 | Local access registers | 4.3.1/4-4 | 0x0_0000: Configuration, control, and status registers<br>0x0_0BF8: Local access window base and size registers |
| 0x0_1000 | e500 coherency module (ECM) | 7.2/7-3 | — |
| 0x0_2000 | DDR memory controller | 8.4/8-10 | — |
| 0x0_3000 | I$^2$C controllers (2) | 9.3/9-9 | I$^2$C controller 1: 0x0_3000<br>I$^2$C controller 2: 0x0_3100 |
| 0x0_4000 | DUART | 12.3/12-3 | UART0: 0x0_4500<br>UART1: 0x0_4600 |
| 0x0_5000 | Enhanced local bus controller (eLBC) | 13.3/13-9 | — |
| 0x0_6000–0x0_6FFF | Reserved | — | — |
| 0x0_7000 | Enhanced serial peripheral interface | 18.3/18-5 | — |
| 0x0_8000 | PCI controller | 16.3/16-11 | — |
| 0x0_9000 | PCI Express controller 2 | 17.3/17-5 | — |
| 0x0_A000 | PCI Express controller 1 | 17.3/17-5 | — |
| 0x0_B000 | PCI Express controller 3 | 17.3/17-5 | — |
| 0x0_C000–0x0_EFFF | Reserved | — | — |
| 0x0_F000 | General-purpose I/O | 22.3/22-2 | — |
| 0x1_0000–0x1_7FFF | Reserved | — | — |
| 0x1_8000 | Serial ATA controller 1 | 19.3.1/19-4 | — |
| 0x1_9000 | Serial ATA controller 2 | 19.3.1/19-4 | — |
| 0x1_A000–0x1_FFFF | Reserved | — | — |
| 0x2_0000 | L2/SRAM memory-mapped configuration | 6.3/6-8 | — |
| 0x2_1000 | DMA controller | 15.3/15-5 | DMA0: 0x2_1100<br>DMA1: 0x2_1180<br>DMA2: 0x2_1200<br>DMA3: 0x2_1280<br>General Status: 0x2_1300 |
| 0x2_2000 | USB controller 1 | 21.3/21-4 | USB3 is at 0x2_B000 |
| 0x2_3000 | USB controller 2 | 21.3/21-4 | — |
| 0x2_4000 | eTSEC1 | 14.5/14-14 | — |
| 0x2_5000 | Reserved | — | — |

### Table 2-11. CCSR Block Base Address Map (continued)

| Block Base Address (Hex) | Block | Section/Page | Comments |
|---|---|---|---|
| 0x2_6000 | eTSEC3 | 14.5/14-14 | — |
| 0x2_7000–<br>0x3_AFFF | Reserved | — | — |
| 0x2_B000 | USB controller 3 | 21.3/21-4 | USB1 is at 0x2_2000;<br>USB2 is at 0x2_3000; |
| 0x2_C000–<br>0x2_DFFF | Reserved | — | — |
| 0x2_E000 | Enhanced secure digital controller | 20.4/20-5 | — |
| 0x2_F000–<br>0x3_1004 | Reserved | — | — |
| 0x3_0000 | Integrated security engine | 10.2/10-11 | — |
| **Programmable Interrupt Controller (PIC) (0x4_0000–0x7_FFFF)** | | | |
| 0x4_0000 | PIC—Global registers | 9.3.1/9-19 | — |
| 0x5_0000 | PIC—Interrupt source configuration registers | 9.3.7/9-40 | — |
| 0x6_0000 | PIC— Per-CPU registers | 9.3.8/9-46 | — |
| 0x7_0000–<br>0xD_FFFF | Reserved | — | — |
| **Device Specific Utilities (0xE_0000–0xF_FFFF)** | | | |
| 0xE_0000 | Global Utilities | 23.4/23-3 | — |
| 0xE_1000 | Performance Monitor | 24.3/24-3 | — |
| 0xE_2000 | Watchpoint Monitor and Trace Buffer | 25.3/25-9 | — |
| 0xE_3000 | SerDes1 (8-lane) control | 23.4/23-3 | — |
| 0xE_3100 | SerDes2 (2-lane) control | 23.4/23-3 | — |
| 0xE_3200–<br>0xE_FFFF | Reserved | — | — |
| 0xF_0000 | Internal Bootrom[1] | — | — |

[1] Even though it is allocated 64 Kbytes in the memory space, only 8 Kbytes of internal bootrom is physically implemented, and this is located at the upper 8 Kbytes of the allocated 64kbyte address space, from CCSR offset 0xF_E000 to 0xF_FFFF.

# Chapter 3
# Signal Descriptions

This chapter describes the MPC8536E external signals. It is organized into the following sections:

- Overview of signals and cross-references for signals that serve multiple functions, including two lists: one by functional block and one alphabetical
- List of reset configuration signals
- List of output signal states at reset

**NOTE**

A bar over a signal name indicates that the signal is active low, such as $\overline{\text{IRQ\_OUT}}$ (interrupt out). Active-low signals are referred to as asserted (active) when they are low and negated when they are high. Signals that are not active low, such as IRQ (interrupt input), are referred to as asserted when they are high and negated when they are low.

Internal signals are shown throughout this document as lower case and in italics. For example, *sys_logic_clk* is an internal signal. These are discussed only as necessary for understanding the external functionality of the device.

## 3.1 Signals Overview

The MPC8536E signals are grouped as follows:

- DDR memory interface signals
- PCI interface signals
- eTSEC 1 and 3 interface signals
- 1588 signals
- Enhanced local bus controller signals
- DMA interface signals
- PIC interface signals
- DUART interface signals
- I$^2$C interface signals
- PCI Express 1, 2, and 3 interface signals
- SATA and SGMII signals
- System control, power management, and debug signals
- Test, JTAG, configuration, and clock signals
- USB interface signals
- eSDHC signals

- SPI signals
- Configuration signals
- General-purpose input/output signals

Note that individual chapters of this document provide details for each signal, describing each signal's behavior when the signal is asserted or negated and when the signal is an input or an output.

The following tables provide summaries of signal functions. Table 3-1 lists the signals grouped by function, and Table 3-2 lists the signals alphabetically. These tables detail the signal name, interface, alternate functions, number of signals, and whether the signal is an input, output, or bidirectional. The direction of the multiplexed signals applies for the primary signal function listed in the left-most column of the table for that row (and does not apply for the state of the reset configuration signals). Finally, the table provides a pointer to the table where the signal function is described.

Table 3-1 provides a summary of the signals grouped by function.

**Table 3-1. MPC8536E Signal Reference by Functional Block**

| Name | Description | Functional Block | Alternate Function(s) | No. of Signals | I/O | Table/ Page |
|---|---|---|---|---|---|---|
| MDQ[0:63] | DDR data | DDR memory | — | 64 | I/O | 8-3/8-6 |
| MECC[0:7] | DDR error correcting code | DDR memory | — | 8 | I/O | 8-3/8-6 |
| MDM[0:7] | DDR data mask | DDR memory | — | 8 | O | 8-3/8-6 |
| MDM8 | DDR ECC data mask | DDR memory | — | 1 | O | 8-3/8-6 |
| MDQS[0:7] | DDR data strobe | DDR memory | — | 8 | I/O | 8-3/8-6 |
| MDQS8 | DDR ECC data strobe | DDR memory | — | 1 | I/O | 8-3/8-6 |
| $\overline{\text{MDQS}}$[0:7] | DDR data strobe (complement) | DDR memory | — | 8 | I/O | 8-3/8-6 |
| $\overline{\text{MDQS8}}$ | DDR ECC data strobe (complement) | DDR memory | — | 1 | I/O | 8-3/8-6 |
| MBA[2:0] | DDR bank select | DDR memory | — | 3 | O | 8-3/8-6 |
| MA[15:0] | DDR address | DDR memory | — | 16 | O | 8-3/8-6 |
| $\overline{\text{MWE}}$ | DDR write enable | DDR memory | — | 1 | O | 8-3/8-6 |
| $\overline{\text{MRAS}}$ | DDR row address strobe | DDR memory | — | 1 | O | 8-3/8-6 |
| $\overline{\text{MCAS}}$ | DDR column address strobe | DDR memory | — | 1 | O | 8-3/8-6 |
| $\overline{\text{MCS}}$[0:3] | DDR chip select (2/DIMM) | DDR memory | — | 4 | O | 8-3/8-6 |
| MCKE[0:3] | DDR clock enable | DDR memory | — | 4 | O | 8-3/8-6 |
| MCK[0:5],<br>$\overline{\text{MCK}}$[0:5] | DDR differential clocks (3 pairs/DIMM) | DDR memory | — | 12 | O | 8-3/8-6 |
| MODT[0:3] | DRAM On-Die Termination | DDR memory | — | 4 | O | 8-3/8-6 |
| MDIC[0:1] | Driver impedance calibration | Debug/ DDR memory | — | 2 | I/O | 8-3/8-6 |
| $\overline{\text{MAPAR\_ERR}}$ | DDR address parity in | DDR memory | — | 1 | I | 8-3/8-6 |
| MAPAR_OUT | DDR address parity out | DDR memory | — | 1 | O | 8-3/8-6 |
| PCI_AD[31:0] | PCI address/data | PCI | — | 32 | I/O | 16-2/16-6 |

### Table 3-1. MPC8536E Signal Reference by Functional Block (continued)

| Name | Description | Functional Block | Alternate Function(s) | No. of Signals | I/O | Table/ Page |
|---|---|---|---|---|---|---|
| PCI_C/BE[3:0] | PCI command/byte enable | PCI | — | 4 | I/O | 16-2/16-6 |
| PCI_PAR | PCI parity | PCI | — | 1 | I/O | 16-2/16-6 |
| PCI_FRAME | PCI frame | PCI | — | 1 | I/O | 16-2/16-6 |
| PCI_TRDY | PCI target ready | PCI | — | 1 | I/O | 16-2/16-6 |
| PCI_IRDY | PCI initiator ready | PCI | — | 1 | I/O | 16-2/16-6 |
| PCI_STOP | PCI stop | PCI | — | 1 | I/O | 16-2/16-6 |
| PCI_DEVSEL | PCI device select | PCI | — | 1 | I/O | 16-2/16-6 |
| PCI_IDSEL | PCI initial device select | PCI | — | 1 | I | 16-2/16-6 |
| PCI_PERR | PCI parity error | PCI | — | 1 | I/O | 16-2/16-6 |
| PCI_SERR | PCI system error | PCI | — | 1 | I/O | 16-2/16-6 |
| PCI_REQ0 | PCI request 0 | PCI | — | 1 | I/O | 16-2/16-6 |
| PCI_REQ[1:2] | PCI request 1–2 | PCI | — | 2 | I | 16-2/16-6 |
| PCI_REQ[3:4] | PCI request 3–4 | PCI | GPIO[0:1] | 2 | I | 16-2/16-6 |
| PCI_GNT0 | PCI grant 0 | PCI | — | 1 | I/O | 16-2/16-6 |
| PCI_GNT1 | PCI grant 1 | PCI | cfg_pci_impd | 1 | O | 16-2/16-6 |
| PCI_GNT2 | PCI grant 2 | PCI | cfg_pci_arb | 1 | O | 16-2/16-6 |
| PCI_GNT[3:4] | PCI grant 3 | PCI | GPIO[2:3] | 1 | O | 16-2/16-6 |
| PCI_CLK | PCI clock | PCI | — | 1 | I | 16-2/16-6 |
| EC_GTX_CLK125 | Gigabit reference clock | Gigabit clock | — | 1 | I | 16-2/16-6 |
| EC_MDC | Ethernet management data clock | Ethernet management | cfg_eng_use0 | 1 | O | 16-2/16-6 |
| EC_MDIO | Ethernet management data in/out | Ethernet management | — | 1 | I/O | 16-2/16-6 |
| TSEC1_TXD[7:4] | TSEC1 transmit data 7–4 | eTSEC1 | FIFO1_TXD[7:4]/ cfg_rom_loc[0:3] | 4 | O | 14-2/14-10 |
| TSEC1_TXD3 | TSEC1 transmit data 3 | eTSEC1 | FIFO1_TXD3/ cfg_eng_use1 | 1 | O | 14-2/14-10 |
| TSEC1_TXD2 | TSEC1 transmit data 2 | eTSEC1 | FIFO1_TXD2/ cfg_srds2_prtcl0 | 1 | O | 14-2/14-10 |
| TSEC1_TXD[1:0] | TSEC1 transmit data 1 | eTSEC1 | FIFO1_TXD1/ cfg_tsec1_prtcl[1:0] | 2 | O | 14-2/14-10 |
| TSEC1_TX_EN | TSEC1 transmit enable | eTSEC1 | FIFO1_TX_EN | 1 | O | 14-2/14-10 |
| TSEC1_TX_ER | TSEC1 transmit error | eTSEC1 | FIFO1_TX_ER/ cfg_tsec1_reduce | 1 | O | 14-2/14-10 |
| TSEC1_TX_CLK | TSEC1 transmit clock in | eTSEC1 | FIFO1_TX_CLK | 1 | I | 14-2/14-10 |
| TSEC1_GTX_CLK | TSEC1 transmit clock out | eTSEC1 | _ | 1 | O | 14-2/14-10 |
| TSEC1_CRS | TSEC1 carrier sense | eTSEC1 | FIFO1_RX_FC | 1 | I/O | 14-2/14-10 |

Table 3-1. MPC8536E Signal Reference by Functional Block (continued)

| Name | Description | Functional Block | Alternate Function(s) | No. of Signals | I/O | Table/ Page |
|---|---|---|---|---|---|---|
| TSEC1_COL | TSEC1 collision detect | eTSEC1 | FIFO1_TX_FC | 1 | I | 14-2/14-10 |
| TSEC1_RXD[7:0] | TSEC1 receive data | eTSEC1 | FIFO1_RXD[7:0] | 8 | I | 14-2/14-10 |
| TSEC1_RX_DV | TSEC1 receive data valid | eTSEC1 | FIFO1_RX_DV | 1 | I | 14-2/14-10 |
| TSEC1_RX_ER | TSEC1 receiver error | eTSEC1 | FIFO1_RX_ER | 1 | I | 14-2/14-10 |
| TSEC1_RX_CLK | TSEC1 receive clock | eTSEC1 | FIFO1_RX_CLK | 1 | I | 14-2/14-10 |
| TSEC3_TXD7 | TSEC3 transmit data 7 | eTSEC3 | FIFO3_TXD7/ cfg_eng_use2 | 1 | O | 14-2/14-10 |
| TSEC3_TXD[6:4] | TSEC3 transmit data [6:4] | eTSEC3 | FIFO3_TXD[6:4]/ cfg_io_ports[0:2] | 3 | O | 14-2/14-10 |
| TSEC3_TXD3 | TSEC3 transmit data 3 | eTSEC3 | FIFO3_TXD3/ cfg_srds2_ref_clk0 | 1 | O | 14-2/14-10 |
| TSEC3_TXD2 | TSEC3 transmit data 2 | eTSEC3 | FIFO3_TXD2/ cfg_srds2_prtcl1 | 1 | O | 14-2/14-10 |
| TSEC3_TXD[1:0] | TSEC3 transmit data 1–0 | eTSEC3 | FIFO3_TXD[1:0]/ cfg_tsec3_prtcl[1:0] | 2 | O | 14-2/14-10 |
| TSEC3_TX_EN | TSEC3 transmit enable | eTSEC3 | FIFO3_TX_EN | 1 | O | 14-2/14-10 |
| TSEC3_TX_ER | TSEC3 transmit error | eTSEC3 | FIFO3_TX_ER/ cfg_tsec3_reduce | 1 | O | 14-2/14-10 |
| TSEC3_TX_CLK | TSEC3 transmit clock in | eTSEC3 | FIFO3_TX_CLK | 1 | I | 14-2/14-10 |
| TSEC3_GTX_CLK | TSEC3 transmit clock out | eTSEC3 | — | 1 | O | 14-2/14-10 |
| TSEC3_CRS | TSEC3 carrier sense | eTSEC3 | FIFO3_RX_FC | 1 | I/O | 14-2/14-10 |
| TSEC3_COL | TSEC3 collision detect | eTSEC3 | FIFO3_TX_FC | 1 | I | 14-2/14-10 |
| TSEC3_RXD[7:0] | TSEC3 receive data 7–0 | eTSEC3 | FIFO3_RXD[7:0] | 8 | I | 14-2/14-10 |
| TSEC3_RX_DV | TSEC3 receive data valid | eTSEC3 | FIFO3_RX_DV | 1 | I | 14-2/14-10 |
| TSEC3_RX_ER | TSEC3 receive error | eTSEC3 | FIFO3_RX_ER | 1 | I | 14-2/14-10 |
| TSEC3_RX_CLK | TSEC3 receive clock | eTSEC3 | FIFO3_RX_CLK | 1 | I | 14-2/14-10 |
| TSEC_1588_CLK | IEEE 1588 clock | IEEE 1588 | — | 1 | I | 14-2/14-10 |
| TSEC_1588_ CLK_OUT | IEEE 1588 clock out | IEEE 1588 | cfg_ddr_pll2 | 1 | O | 14-2/14-10 |
| TSEC_1588_ TRIG_IN[0:1] | IEEE 1588 trigger in | IEEE 1588 | | 2 | I | 14-2/14-10 |
| TSEC_1588_ PULSE_OUT1 | IEEE 1588 pulse out 1 | IEEE 1588 | cfg_srds2_prtcl2 | 1 | O | 14-2/14-10 |
| TSEC_1588_ PULSE_OUT2 | IEEE 1588 pulse out 2 | IEEE 1588 | cfg_srds2_ref_clk1 | 1 | O | 14-2/14-10 |
| TSEC_1588_ TRIG_OUT[0:1] | IEEE 1588 alarm out | IEEE 1588 | cfg_ddr_pll[0:1] | 2 | O | 14-2/14-10 |
| LAD[0:31] | Local bus address/data | eLBC | cfg_gpinput[0:31] | 32 | I/O | 13-2/13-5 |
| LDP[0:3] | Local bus data parity | eLBC | — | 4 | I/O | 13-2/13-5 |

Table 3-1. MPC8536E Signal Reference by Functional Block (continued)

| Name | Description | Functional Block | Alternate Function(s) | No. of Signals | I/O | Table/ Page |
|---|---|---|---|---|---|---|
| LA27 | Local bus burst address | eLBC | cfg_cpu_boot | 1 | O | 13-2/13-5 |
| LA[28:31] | Local bus port address | eLBC | cfg_sys_pll[0:3] | 4 | O | 13-2/13-5 |
| $\overline{\text{LCS}}$[0:4] | Local bus chip select 0–4 | eLBC | — | 5 | O | 13-2/13-5 |
| $\overline{\text{LCS5}}$ | Local bus chip select 5 | eLBC | $\overline{\text{DMA\_DREQ2}}$ | 1 | I/O | 13-2/13-5 |
| $\overline{\text{LCS6}}$ | Local bus chip select 6 | eLBC | $\overline{\text{DMA\_DACK2}}$ | 1 | O | 13-2/13-5 |
| $\overline{\text{LCS7}}$ | Local bus chip select 7 | eLBC | $\overline{\text{DMA\_DDONE2}}$ | 1 | O | 13-2/13-5 |
| $\overline{\text{LWE0}}$/<br>$\overline{\text{LBS0}}$/$\overline{\text{LFWE}}$ | Local bus write enable /byte select 0 | eLBC | cfg_core_speed | 1 | O | 13-2/13-5 |
| $\overline{\text{LWE}}$[1:3]/<br>$\overline{\text{LBS}}$[1:3] | Local bus write enable /byte select 1–3 | eLBC | cfg_host_agt[0:2] | 3 | O | 13-2/13-5 |
| LBCTL | Local bus data buffer control | eLBC | cfg_core_pll0 | 1 | O | 13-2/13-5 |
| LALE | Local bus address latch enable | eLBC | cfg_core_pll1 | 1 | O | 13-2/13-5 |
| LGPL0/LFCLE | Local bus UPM general purpose line 0 | eLBC | cfg_dram_type | 1 | O | 13-2/13-5 |
| LGPL1/LFALE | Local bus GP line 1 | eLBC | cfg_sys_speed | 1 | O | 13-2/13-5 |
| LGPL2/<br>$\overline{\text{LOE}}$/$\overline{\text{LFRE}}$ | Local bus GP line 2 /output enable | eLBC | cfg_core_pll2 | 1 | O | 13-2/13-5 |
| LGPL3/$\overline{\text{LFWP}}$ | Local bus GP line 3 | eLBC | cfg_boot_seq0 | 1 | O | 13-2/13-5 |
| LGPL4/$\overline{\text{LGTA}}$/<br>LFR$\overline{\text{B}}$/LUPWAIT/<br>LPBSE | Local bus GP line 4/GPCM terminate access/UPM wait | eLBC | — | 1 | I/O | 13-2/13-5 |
| LGPL5 | Local bus GP line 5 address | eLBC | cfg_boot_seq1 | 1 | O | 13-2/13-5 |
| LCLK[0:2] | Local bus clock | eLBC | — | 3 | O | 13-2/13-5 |
| LSYNC_IN | Local bus PLL synchronization | eLBC | — | 1 | I | 13-2/13-5 |
| LSYNC_OUT | Local bus PLL synchronization | eLBC | — | 1 | O | 13-2/13-5 |
| $\overline{\text{DMA\_DREQ}}$[0:1] | DMA request 0–1 | DMA | GPIO[14:15] | 2 | I | 15-3/15-5 |
| $\overline{\text{DMA\_DREQ2}}$ | DMA request 2 | DMA | $\overline{\text{LCS5}}$ | 1 | I | 15-3/15-5 |
| $\overline{\text{DMA\_DREQ3}}$ | DMA request 3 | DMA | IRQ9 | 1 | I | 15-3/15-5 |
| $\overline{\text{DMA\_DACK}}$[0:1] | DMA acknowledge 0–1 | DMA | GPIO[10:11] | 2 | O | 15-3/15-5 |
| $\overline{\text{DMA\_DACK2}}$ | DMA acknowledge 2 | DMA | $\overline{\text{LCS6}}$ | 1 | O | 15-3/15-5 |
| $\overline{\text{DMA\_DACK3}}$ | DMA acknowledge 3 | DMA | IRQ10 | 1 | O | 15-3/15-5 |
| $\overline{\text{DMA\_DDONE}}$[0:1] | DMA done 0–1 | DMA | GPIO[12:13] | 2 | O | 15-3/15-5 |
| $\overline{\text{DMA\_DDONE2}}$ | DMA done 2 | DMA | $\overline{\text{LCS7}}$ | 1 | O | 15-3/15-5 |
| $\overline{\text{DMA\_DDONE3}}$ | DMA done 3 | DMA | IRQ11 | 1 | O | 15-3/15-5 |
| $\overline{\text{MCP}}$ | Machine check processor | PIC | — | 1 | I | 9-3/9-8 |
| $\overline{\text{UDE}}$ | Unconditional debug event | PIC | — | 1 | I | 9-3/9-8 |
| IRQ[0:8] | External interrupt 0–8 | PIC | — | 9 | I | 9-3/9-8 |

Table 3-1. MPC8536E Signal Reference by Functional Block (continued)

| Name | Description | Functional Block | Alternate Function(s) | No. of Signals | I/O | Table/ Page |
|---|---|---|---|---|---|---|
| IRQ9 | External interrupt 9 | PIC | $\overline{\text{DMA\_DREQ3}}$ | 1 | I | 9-3/9-8 |
| IRQ10 | External interrupt 10 | PIC | $\overline{\text{DMA\_DACK3}}$ | 1 | I | 9-3/9-8 |
| IRQ11 | External interrupt 11 | PIC | $\overline{\text{DMA\_DDONE3}}$ | 1 | I | 9-3/9-8 |
| $\overline{\text{IRQ\_OUT}}$ | Interrupt output | PIC | — | 1 | O | 9-3/9-8 |
| UART_SIN[0:1] | DUART serial data in | Dual UART | — | 2 | I | 12-1/12-3 |
| UART_SOUT[0:1] | DUART serial data out | Dual UART | — | 2 | O | 12-1/12-3 |
| $\overline{\text{UART\_CTS}}$[0:1] | DUART clear to send | Dual UART | — | 2 | I | 12-1/12-3 |
| $\overline{\text{UART\_RTS}}$[0:1] | DUART ready to send | Dual UART | — | 2 | O | 12-1/12-3 |
| IIC1_SDA | I$^2$C serial data | I$^2$C | — | 1 | I/O | 11-2/11-4 |
| IIC1_SCL | I$^2$C serial clock | I$^2$C | — | 1 | I/O | 11-2/11-4 |
| IIC2_SDA | I$^2$C serial data | I$^2$C | — | 1 | I/O | 11-2/11-4 |
| IIC2_SCL | I$^2$C serial clock | I$^2$C | — | 1 | I/O | 11-2/11-4 |
| SD1_RX[7:6],<br>$\overline{\text{SD1\_RX}}$[7:6] | PCI Express receive data, receive data complement | PCI Express 1/ PCI Express 2/ PCI Express 3 | — | 2 | I | 17-2/17-5 |
| SD1_RX[5:4],<br>$\overline{\text{SD1\_RX}}$[5:4] | PCI Express receive data, receive data complement | PCI Express 1/ PCI Express 2 | — | 2 | I | 17-2/17-5 |
| SD1_RX[3:0],<br>$\overline{\text{SD1\_RX}}$[3:0] | PCI Express receive data, receive data complement | PCI Express 1 | — | 4 | I | 17-2/17-5 |
| SD1_TX[7:6],<br>$\overline{\text{SD1\_TX}}$[7:6] | PCI Express transmit data, transmit data complement | PCI Express 1/ PCI Express 2/ PCI Express 3 | — | 2 | O | 17-2/17-5 |
| SD1_TX[5:4],<br>$\overline{\text{SD1\_TX}}$[5:4] | PCI Express transmit data, transmit data complement | PCI Express 1/ PCI Express 2 | — | 2 | O | 17-2/17-5 |
| SD1_TX[3:0],<br>$\overline{\text{SD1\_TX}}$[3:0] | PCI Express transmit data, transmit data complement | PCI Express 1 | — | 4 | O | 17-2/17-5 |
| SD1_REF_CLK,<br>$\overline{\text{SD1\_REF\_CLK}}$ | SerDes1 reference clock, SerDes1 reference clock complement | PCI Express 1, PCI Express 2, PCI Express 3 | — | 2 | I | |
| SD2_RX[1:0],<br>$\overline{\text{SD2\_RX}}$[1:0] | Receive data, receive data complement | SATA, SGMII | — | 4 | I | |
| SD2_TX[1:0],<br>$\overline{\text{SD2\_TX}}$[1:0] | Transmit data, transmit data complement | SATA, SGMII | — | 4 | O | |
| SD2_REF_CLK,<br>$\overline{\text{SD2\_REF\_CLK}}$ | SerDes2 reference clock, SerDes2 reference clock complement | SATA, SGMII | — | 2 | I | |
| $\overline{\text{HRESET}}$ | Hard reset | System control | — | 1 | I | 4-2/4-2 |
| $\overline{\text{HRESET\_REQ}}$ | Hard reset request | System control | — | 1 | O | 4-2/4-2 |
| $\overline{\text{SRESET}}$ | Soft reset | System control | — | 1 | I | 4-2/4-2 |
| $\overline{\text{CKSTP\_IN}}$ | Checkstop in | System control | — | 1 | I | 23-2/23-2 |

Table 3-1. MPC8536E Signal Reference by Functional Block (continued)

| Name | Description | Functional Block | Alternate Function(s) | No. of Signals | I/O | Table/ Page |
|---|---|---|---|---|---|---|
| $\overline{\text{CKSTP\_OUT}}$ | Checkstop out | System control | — | 1 | O | 23-2/23-2 |
| READY | Device ready | System control | TRIG_OUT | 1 | O | 4-2/4-2 |
| ASLEEP | Asleep | Power mgmt | — | 1 | O | 23-2/23-2 |
| POWER_OK | Stable power | Power mgmt | — | 1 | I | 23-2/23-2 |
| POWER_EN | Power enable | Power mgmt | — | 1 | O | 23-2/23-2 |
| TRIG_IN | Watchpoint trigger in | Debug | — | 1 | I | 25-4/25-7 |
| TRIG_OUT | Watchpoint trigger out | Debug | READY | 1 | O | 25-4/25-7 |
| MSRCID0 | Memory debug source port ID 0 | Debug | cfg_mem_debug | 1 | O | 25-3/25-6 |
| MSRCID1 | Memory debug source port ID 1 | Debug | cfg_ddr_debug | 1 | O | 25-3/25-6 |
| MSRCID[2:4] | Memory debug source port ID 2–4 | Debug | — | 3 | O | 25-3/25-6 |
| MDVAL | Memory debug data valid | Debug | — | 1 | O | 25-3/25-6 |
| $\overline{\text{LSSD\_MODE}}$ | LSSD mode | Test | — | 1 | I | 25-5/25-8 |
| L1_TSTCLK | L1 test clock | Test | — | 1 | I | 25-5/25-8 |
| L2_TSTCLK | L2 test clock | Test | — | 1 | I | 25-5/25-8 |
| $\overline{\text{TEST\_SEL}}$ | Test select | Test | — | 1 | I | 25-5/25-8 |
| TEMP_ANODE | Thermal diode access | Test | — | 2 | I | 25-5/25-8 |
| TEMP_CATHODE | Thermal diode access | Test | — | 2 | I | 25-5/25-8 |
| TCK | Test clock | JTAG | — | 1 | I | 25-5/25-8 |
| TDI | Test data in | JTAG | — | 1 | I | 25-5/25-8 |
| TDO | Test data out | JTAG | — | 1 | O | 25-5/25-8 |
| TMS | Test mode select | JTAG | — | 1 | I | 25-5/25-8 |
| $\overline{\text{TRST}}$ | Test reset | JTAG | — | 1 | I | 25-5/25-8 |
| SYSCLK | System clock/PCI clock | Clock | — | 1 | I | 4-3/4-3 |
| RTC | Real time clock | Clock | — | 1 | I | 4-3/4-3 |
| CLK_OUT | Clock out | Clock | — | 1 | O | 23-2/23-2 |
| USB1_D[7:0] | USB1 data | USB1 | — | 8 | I/O | 21-1/21-3 |
| USB1_NXT | USB1 next data | USB1 | — | 1 | I | 21-1/21-3 |
| USB1_DIR | USB1 data | USB1 | — | 1 | I | 21-1/21-3 |
| USB1_STP | USB1 stop | USB1 | cfg_pci_clk | 1 | O | 21-1/21-3 |
| USB1_PWRFAULT | USB1 power fault | USB1 | — | 1 | I | 21-1/21-3 |
| USB1_PCTL0 | USB1 port control 0 | USB1 | GPIO6 | 1 | O | 21-1/21-3 |
| USB1_PCTL1 | USB1 port control 1 | USB1 | GPIO7 | 1 | O | 21-1/21-3 |
| USB1_CLK | USB1 clock | USB1 | — | 1 | I | 21-1/21-3 |
| USB2_D[7:0] | USB2 data | USB2 | — | 8 | I/O | 21-1/21-3 |

### Table 3-1. MPC8536E Signal Reference by Functional Block (continued)

| Name | Description | Functional Block | Alternate Function(s) | No. of Signals | I/O | Table/ Page |
|---|---|---|---|---|---|---|
| USB2_NXT | USB2 next data | USB2 | — | 1 | I | 21-1/21-3 |
| USB2_DIR | USB2 data | USB2 | — | 1 | I | 21-1/21-3 |
| USB2_STP | USB2 stop | USB2 | cfg_pci_speed | 1 | O | 21-1/21-3 |
| USB2_PWRFAULT | USB2 power fault | USB2 | — | 1 | I | 21-1/21-3 |
| USB2_PCTL0 | USB2 port control 0 | USB2 | GPIO8 | 1 | O | 21-1/21-3 |
| USB2_PCTL1 | USB2 port control 1 | USB2 | GPIO9 | 1 | O | 21-1/21-3 |
| USB2_CLK | USB2 clock | USB2 | — | 1 | I | 21-1/21-3 |
| USB3_D[7:0] | USB3 data | USB3 | — | 8 | I/O | 21-1/21-3 |
| USB3_NXT | USB3 next data | USB3 | — | 1 | I | 21-1/21-3 |
| USB3_DIR | USB3 data | USB3 | — | 1 | I | 21-1/21-3 |
| USB3_STP | USB3 stop | USB3 | — | 1 | O | 21-1/21-3 |
| USB3_CLK | USB3 clock | USB3 | — | 1 | I | 21-1/21-3 |
| SDHC_CMD | eSDHC command | eSDHC | — | 1 | I/O | 20-1/20-4 |
| $\overline{\text{SDHC\_CD}}$ | eSDHC | eSDHC | GPIO4 | 1 | I | 20-1/20-4 |
| SDHC_DAT[0:3] | eSDHC data | eSDHC | — | 1 | I/O | 20-1/20-4 |
| SDHC_DAT[4:7] | eSDHC data | eSDHC | $\overline{\text{SPI\_CS}}$[0:3] | 4 | I/O | 20-1/20-4 |
| SDHC_CLK | eSDHC clock | eSDHC | — | 1 | O | 20-1/20-4 |
| SDHC_WP | eSDHC write protect | eSDHC | GPIO5 | 1 | I | 20-1/20-4 |
| SPI_MOSI | SPI master out slave in | SPI | — | 1 | I/O | 18-1/18-4 |
| SPI_MISO | SPI master in slave out | SPI | — | 1 | I | 18-1/18-4 |
| SPI_CLK | SPI clock | SPI | — | 1 | O | 18-1/18-4 |
| $\overline{\text{SPI\_CS}}$[0:3] | SPI chip select 0–3 | SPI | SDHC_DAT[4:7] | 4 | I/O | 18-1/18-4 |
| GPIO[0:1] | General-purpose I/O | GPIO | $\overline{\text{PCI\_REQ}}$[3:4] | 2 | I/O | 22-1/22-2 |
| GPIO2 | General-purpose I/O | GPIO | $\overline{\text{PCI\_GNT3}}$ | 1 | I/O | 22-1/22-2 |
| GPIO3 | General-purpose I/O | GPIO | $\overline{\text{PCI\_GNT4}}$ | 1 | I/O | 22-1/22-2 |
| GPIO4 | General-purpose I/O | GPIO | $\overline{\text{SDHC\_CD}}$ | 1 | I/O | 22-1/22-2 |
| GPIO5 | General-purpose I/O | GPIO | SDHC_WP | 1 | I/O | 22-1/22-2 |
| GPIO6 | General-purpose I/O | GPIO | USB1_PCTL0 | 1 | I/O | 22-1/22-2 |
| GPIO7 | General-purpose I/O | GPIO | USB1_PCTL1 | 1 | I/O | 22-1/22-2 |
| GPIO8 | General-purpose I/O | GPIO | USB2_PCTL0 | 1 | I/O | 22-1/22-2 |
| GPIO9 | General-purpose I/O | GPIO | USB2_PCTL1 | 1 | I/O | 22-1/22-2 |
| GPIO[10:11] | General-purpose I/O | GPIO | $\overline{\text{DMA\_DACK}}$[0:1] | 2 | I/O | 22-1/22-2 |
| GPIO[12:13] | General-purpose I/O | GPIO | $\overline{\text{DMA\_DDONE}}$[0:1] | 2 | I/O | 22-1/22-2 |
| GPIO[14:15] | General-purpose I/O | GPIO | $\overline{\text{DMA\_DREQ}}$[0:1] | 2 | I/O | 22-1/22-2 |

Table 3-2 provides the alphabetical summary list of signals.

Table 3-2. MPC8536E Alphabetical Signal Reference

| Name | Description | Functional Block | Alternate Function(s) | No. of Signals | I/O | Table/ Page |
|---|---|---|---|---|---|---|
| ASLEEP | Asleep | Power mgmt | — | 1 | O | 23-2/23-2 |
| CKSTP_IN | Checkstop in | System control | — | 1 | I | 23-2/23-2 |
| CKSTP_OUT | Checkstop out | System control | — | 1 | O | 23-2/23-2 |
| CLK_OUT | Clock out | Clock | — | 1 | O | 23-2/23-2 |
| DMA_DACK[0:1] | DMA acknowledge 0–1 | DMA | GPIO[10:11] | 2 | O | 15-3/15-5 |
| DMA_DACK2 | DMA acknowledge 2 | DMA | LCS6 | 1 | O | 15-3/15-5 |
| DMA_DACK3 | DMA acknowledge 3 | DMA | IRQ10 | 1 | O | 15-3/15-5 |
| DMA_DDONE[0:1] | DMA done 0–1 | DMA | GPIO[12:13] | 2 | O | 15-3/15-5 |
| DMA_DDONE2 | DMA done 2 | DMA | LCS7 | 1 | O | 15-3/15-5 |
| DMA_DDONE3 | DMA done 3 | DMA | IRQ11 | 1 | O | 15-3/15-5 |
| DMA_DREQ[0:1] | DMA request 0–1 | DMA | GPIO[14:15] | 2 | I | 15-3/15-5 |
| DMA_DREQ2 | DMA request 2 | DMA | LCS5 | 1 | I | 15-3/15-5 |
| DMA_DREQ3 | DMA request 3 | DMA | IRQ9 | 1 | I | 15-3/15-5 |
| EC_GTX_CLK125 | Gigabit reference clock | Gigabit clock | — | 1 | I | 16-2/16-6 |
| EC_MDC | Ethernet management data clock | Ethernet management | cfg_eng_use0 | 1 | O | 16-2/16-6 |
| EC_MDIO | Ethernet management data in/out | Ethernet management | — | 1 | I/O | 16-2/16-6 |
| GPIO[0:1] | General-purpose I/O | GPIO | PCI_REQ[3:4] | 2 | I/O | 22-1/22-2 |
| GPIO[10:11] | General-purpose I/O | GPIO | DMA_DACK[0:1] | 2 | I/O | 22-1/22-2 |
| GPIO[12:13] | General-purpose I/O | GPIO | DMA_DDONE[0:1] | 2 | I/O | 22-1/22-2 |
| GPIO[14:15] | General-purpose I/O | GPIO | DMA_DREQ[0:1] | 2 | I/O | 22-1/22-2 |
| GPIO2 | General-purpose I/O | GPIO | PCI_GNT3 | 1 | I/O | 22-1/22-2 |
| GPIO3 | General-purpose I/O | GPIO | PCI_GNT4 | 1 | I/O | 22-1/22-2 |
| GPIO4 | General-purpose I/O | GPIO | SDHC_CD | 1 | I/O | 22-1/22-2 |
| GPIO5 | General-purpose I/O | GPIO | SDHC_WP | 1 | I/O | 22-1/22-2 |
| GPIO6 | General-purpose I/O | GPIO | USB1_PCTL0 | 1 | I/O | 22-1/22-2 |
| GPIO7 | General-purpose I/O | GPIO | USB1_PCTL1 | 1 | I/O | 22-1/22-2 |
| GPIO8 | General-purpose I/O | GPIO | USB2_PCTL0 | 1 | I/O | 22-1/22-2 |
| GPIO9 | General-purpose I/O | GPIO | USB2_PCTL1 | 1 | I/O | 22-1/22-2 |
| HRESET | Hard reset | System control | — | 1 | I | 4-2/4-2 |
| HRESET_REQ | Hard reset request | System control | — | 1 | O | 4-2/4-2 |
| IIC1_SCL | I$^2$C serial clock | I$^2$C | — | 1 | I/O | 11-2/11-4 |
| IIC1_SDA | I$^2$C serial data | I$^2$C | — | 1 | I/O | 11-2/11-4 |
| IIC2_SCL | I$^2$C serial clock | I$^2$C | — | 1 | I/O | 11-2/11-4 |

### Table 3-2. MPC8536E Alphabetical Signal Reference (continued)

| Name | Description | Functional Block | Alternate Function(s) | No. of Signals | I/O | Table/ Page |
|---|---|---|---|---|---|---|
| IIC2_SDA | I$^2$C serial data | I$^2$C | — | 1 | I/O | 11-2/11-4 |
| IRQ[0:8] | External interrupt 0–8 | PIC | — | 9 | I | 9-3/9-8 |
| IRQ_OUT | Interrupt output | PIC | — | 1 | O | 9-3/9-8 |
| IRQ10 | External interrupt 10 | PIC | DMA_DACK3 | 1 | I | 9-3/9-8 |
| IRQ11 | External interrupt 11 | PIC | DMA_DDONE3 | 1 | I | 9-3/9-8 |
| IRQ9 | External interrupt 9 | PIC | DMA_DREQ3 | 1 | I | 9-3/9-8 |
| L1_TSTCLK | L1 test clock | Test | — | 1 | I | 25-5/25-8 |
| L2_TSTCLK | L2 test clock | Test | — | 1 | I | 25-5/25-8 |
| LA[28:31] | Local bus port address | eLBC | cfg_sys_pll[0:3] | 4 | O | 13-2/13-5 |
| LA27 | Local bus burst address | eLBC | cfg_cpu_boot | 1 | O | 13-2/13-5 |
| LAD[0:31] | Local bus address/data | eLBC | cfg_gpinput[0:31] | 32 | I/O | 13-2/13-5 |
| LALE | Local bus address latch enable | eLBC | cfg_core_pll1 | 1 | O | 13-2/13-5 |
| LBCTL | Local bus data buffer control | eLBC | cfg_core_pll0 | 1 | O | 13-2/13-5 |
| LCLK[0:2] | Local bus clock | eLBC | — | 3 | O | 13-2/13-5 |
| LCS[0:4] | Local bus chip select 0–4 | eLBC | — | 5 | O | 13-2/13-5 |
| LCS5 | Local bus chip select 5 | eLBC | DMA_DREQ2 | 1 | I/O | 13-2/13-5 |
| LCS6 | Local bus chip select 6 | eLBC | DMA_DACK2 | 1 | O | 13-2/13-5 |
| LCS7 | Local bus chip select 7 | eLBC | DMA_DDONE2 | 1 | O | 13-2/13-5 |
| LDP[0:3] | Local bus data parity | eLBC | — | 4 | I/O | 13-2/13-5 |
| LGPL0/LFCLE | Local bus UPM general purpose line 0 | eLBC | cfg_dram_type | 1 | O | 13-2/13-5 |
| LGPL1/LFALE | Local bus GP line 1 | eLBC | cfg_sys_speed | 1 | O | 13-2/13-5 |
| LGPL2/ LOE/LFRE | Local bus GP line 2 /output enable | eLBC | cfg_core_pll2 | 1 | O | 13-2/13-5 |
| LGPL3/LFWP | Local bus GP line 3 | eLBC | cfg_boot_seq0 | 1 | O | 13-2/13-5 |
| LGPL4/LGTA/ LFRB/LUPWAIT/ LPBSE | Local bus GP line 4/GPCM terminate access/UPM wait | eLBC | — | 1 | I/O | 13-2/13-5 |
| LGPL5 | Local bus GP line 5 address | eLBC | cfg_boot_seq1 | 1 | O | 13-2/13-5 |
| LSSD_MODE | LSSD mode | Test | — | 1 | I | 25-5/25-8 |
| LSYNC_IN | Local bus PLL synchronization | eLBC | — | 1 | I | 13-2/13-5 |
| LSYNC_OUT | Local bus PLL synchronization | eLBC | — | 1 | O | 13-2/13-5 |
| LWE[1:3]/ LBS[1:3] | Local bus write enable /byte select 1–3 | eLBC | cfg_host_agt[0:2] | 3 | O | 13-2/13-5 |
| LWE0/ LBS0/LFWE | Local bus write enable /byte select 0 | eLBC | cfg_core_speed | 1 | O | 13-2/13-5 |
| MA[15:0] | DDR address | DDR memory | — | 16 | O | 8-3/8-6 |

### Table 3-2. MPC8536E Alphabetical Signal Reference (continued)

| Name | Description | Functional Block | Alternate Function(s) | No. of Signals | I/O | Table/ Page |
|---|---|---|---|---|---|---|
| $\overline{\text{MAPAR\_ERR}}$ | DDR address parity in | DDR memory | — | 1 | I | 8-3/8-6 |
| MAPAR_OUT | DDR address parity out | DDR memory | — | 1 | O | 8-3/8-6 |
| MBA[2:0] | DDR bank select | DDR memory | — | 3 | O | 8-3/8-6 |
| $\overline{\text{MCAS}}$ | DDR column address strobe | DDR memory | — | 1 | O | 8-3/8-6 |
| MCK[0:5],<br>$\overline{\text{MCK}}$[0:5] | DDR differential clocks (3 pairs/DIMM) | DDR memory | — | 12 | O | 8-3/8-6 |
| MCKE[0:3] | DDR clock enable | DDR memory | — | 4 | O | 8-3/8-6 |
| $\overline{\text{MCP}}$ | Machine check processor | PIC | — | 1 | I | 9-3/9-8 |
| $\overline{\text{MCS}}$[0:3] | DDR chip select (2/DIMM) | DDR memory | — | 4 | O | 8-3/8-6 |
| MDIC[0:1] | Driver impedance calibration | Debug/ DDR memory | — | 2 | I/O | 8-3/8-6 |
| MDM[0:7] | DDR data mask | DDR memory | — | 8 | O | 8-3/8-6 |
| MDM8 | DDR ECC data mask | DDR memory | — | 1 | O | 8-3/8-6 |
| MDQ[0:63] | DDR data | DDR memory | — | 64 | I/O | 8-3/8-6 |
| MDQS[0:7] | DDR data strobe | DDR memory | — | 8 | I/O | 8-3/8-6 |
| $\overline{\text{MDQS}}$[0:7] | DDR data strobe (complement) | DDR memory | — | 8 | I/O | 8-3/8-6 |
| MDQS8 | DDR ECC data strobe | DDR memory | — | 1 | I/O | 8-3/8-6 |
| $\overline{\text{MDQS8}}$ | DDR ECC data strobe (complement) | DDR memory | — | 1 | I/O | 8-3/8-6 |
| MDVAL | Memory debug data valid | Debug | — | 1 | O | 25-3/25-6 |
| MECC[0:7] | DDR error correcting code | DDR memory | — | 8 | I/O | 8-3/8-6 |
| MODT[0:3] | DRAM On-Die Termination | DDR memory | — | 4 | O | 8-3/8-6 |
| $\overline{\text{MRAS}}$ | DDR row address strobe | DDR memory | — | 1 | O | 8-3/8-6 |
| MSRCID[2:4] | Memory debug source port ID 2–4 | Debug | — | 3 | O | 25-3/25-6 |
| MSRCID0 | Memory debug source port ID 0 | Debug | cfg_mem_debug | 1 | O | 25-3/25-6 |
| MSRCID1 | Memory debug source port ID 1 | Debug | cfg_ddr_debug | 1 | O | 25-3/25-6 |
| $\overline{\text{MWE}}$ | DDR write enable | DDR memory | — | 1 | O | 8-3/8-6 |
| PCI_AD[31:0] | PCI address/data | PCI | — | 32 | I/O | 16-2/16-6 |
| PCI_C/$\overline{\text{BE}}$[3:0] | PCI command/byte enable | PCI | — | 4 | I/O | 16-2/16-6 |
| PCI_CLK | PCI clock | PCI | — | 1 | I | 16-2/16-6 |
| $\overline{\text{PCI\_DEVSEL}}$ | PCI device select | PCI | — | 1 | I/O | 16-2/16-6 |
| $\overline{\text{PCI\_FRAME}}$ | PCI frame | PCI | — | 1 | I/O | 16-2/16-6 |
| $\overline{\text{PCI\_GNT}}$[3:4] | PCI grant 3 | PCI | GPIO[2:3] | 1 | O | 16-2/16-6 |
| $\overline{\text{PCI\_GNT0}}$ | PCI grant 0 | PCI | — | 1 | I/O | 16-2/16-6 |
| $\overline{\text{PCI\_GNT1}}$ | PCI grant 1 | PCI | cfg_pci_impd | 1 | O | 16-2/16-6 |

### Table 3-2. MPC8536E Alphabetical Signal Reference (continued)

| Name | Description | Functional Block | Alternate Function(s) | No. of Signals | I/O | Table/ Page |
|---|---|---|---|---|---|---|
| $\overline{\text{PCI\_GNT2}}$ | PCI grant 2 | PCI | cfg_pci_arb | 1 | O | 16-2/16-6 |
| PCI_IDSEL | PCI initial device select | PCI | — | 1 | I | 16-2/16-6 |
| $\overline{\text{PCI\_IRDY}}$ | PCI initiator ready | PCI | — | 1 | I/O | 16-2/16-6 |
| PCI_PAR | PCI parity | PCI | — | 1 | I/O | 16-2/16-6 |
| $\overline{\text{PCI\_PERR}}$ | PCI parity error | PCI | — | 1 | I/O | 16-2/16-6 |
| $\overline{\text{PCI\_REQ}}$[1:2] | PCI request 1–2 | PCI | — | 2 | I | 16-2/16-6 |
| $\overline{\text{PCI\_REQ}}$[3:4] | PCI request 3–4 | PCI | GPIO[0:1] | 2 | I | 16-2/16-6 |
| $\overline{\text{PCI\_REQ0}}$ | PCI request 0 | PCI | — | 1 | I/O | 16-2/16-6 |
| $\overline{\text{PCI\_SERR}}$ | PCI system error | PCI | — | 1 | I/O | 16-2/16-6 |
| $\overline{\text{PCI\_STOP}}$ | PCI stop | PCI | — | 1 | I/O | 16-2/16-6 |
| $\overline{\text{PCI\_TRDY}}$ | PCI target ready | PCI | — | 1 | I/O | 16-2/16-6 |
| POWER_EN | Power enable | Power mgmt | — | 1 | O | 23-2/23-2 |
| POWER_OK | Stable power | Power mgmt | — | 1 | I | 23-2/23-2 |
| READY | Device ready | System control | TRIG_OUT | 1 | O | 4-2/4-2 |
| RTC | Real time clock | Clock | — | 1 | I | 4-3/4-3 |
| SD1_REF_CLK,<br>$\overline{\text{SD1\_REF\_CLK}}$ | SerDes1 reference clock, SerDes1 reference clock complement | PCI Express 1, PCI Express 2, PCI Express 3 | — | 2 | I | |
| SD1_RX[3:0],<br>$\overline{\text{SD1\_RX}}$[3:0] | PCI Express receive data, receive data complement | PCI Express 1 | — | 4 | I | 17-2/17-5 |
| SD1_RX[5:4],<br>$\overline{\text{SD1\_RX}}$[5:4] | PCI Express receive data, receive data complement | PCI Express 1/ PCI Express 2 | — | 2 | I | 17-2/17-5 |
| SD1_RX[7:6],<br>$\overline{\text{SD1\_RX}}$[7:6] | PCI Express receive data, receive data complement | PCI Express 1/ PCI Express 2/ PCI Express 3 | — | 2 | I | 17-2/17-5 |
| SD1_TX[3:0],<br>$\overline{\text{SD1\_TX}}$[3:0] | PCI Express transmit data, transmit data complement | PCI Express 1 | — | 4 | O | 17-2/17-5 |
| SD1_TX[5:4],<br>$\overline{\text{SD1\_TX}}$[5:4] | PCI Express transmit data, transmit data complement | PCI Express 1/ PCI Express 2 | — | 2 | O | 17-2/17-5 |
| SD1_TX[7:6],<br>$\overline{\text{SD1\_TX}}$[7:6] | PCI Express transmit data, transmit data complement | PCI Express 1/ PCI Express 2/ PCI Express 3 | — | 2 | O | 17-2/17-5 |
| SD2_REF_CLK,<br>$\overline{\text{SD2\_REF\_CLK}}$ | SerDes2 reference clock, SerDes2 reference clock complement | SATA, SGMII | — | 2 | I | |
| SD2_RX[1:0],<br>$\overline{\text{SD2\_RX}}$[1:0] | Receive data, receive data complement | SATA, SGMII | — | 4 | I | |
| SD2_TX[1:0],<br>$\overline{\text{SD2\_TX}}$[1:0] | Transmit data, transmit data complement | SATA, SGMII | — | 4 | O | |
| $\overline{\text{SDHC\_CD}}$ | eSDHC | eSDHC | GPIO4 | 1 | I | 20-1/20-4 |

### Table 3-2. MPC8536E Alphabetical Signal Reference (continued)

| Name | Description | Functional Block | Alternate Function(s) | No. of Signals | I/O | Table/ Page |
|---|---|---|---|---|---|---|
| SDHC_CLK | eSDHC clock | eSDHC | — | 1 | O | 20-1/20-4 |
| SDHC_CMD | eSDHC command | eSDHC | — | 1 | I/O | 20-1/20-4 |
| SDHC_DAT[0:3] | eSDHC data | eSDHC | — | 1 | I/O | 20-1/20-4 |
| SDHC_DAT[4:7] | eSDHC data | eSDHC | SPI_CS[0:3] | 4 | I/O | 20-1/20-4 |
| SDHC_WP | eSDHC write protect | eSDHC | GPIO5 | 1 | I | 20-1/20-4 |
| SPI_CLK | SPI clock | SPI | — | 1 | O | 18-1/18-4 |
| SPI_CS[0:3] | SPI chip select 0–3 | SPI | SDHC_DAT[4:7] | 4 | I/O | 18-1/18-4 |
| SPI_MISO | SPI master in slave out | SPI | — | 1 | I | 18-1/18-4 |
| SPI_MOSI | SPI master out slave in | SPI | — | 1 | I/O | 18-1/18-4 |
| SRESET | Soft reset | System control | — | 1 | I | 4-2/4-2 |
| SYSCLK | System clock/PCI clock | Clock | — | 1 | I | 4-3/4-3 |
| TCK | Test clock | JTAG | — | 1 | I | 25-5/25-8 |
| TDI | Test data in | JTAG | — | 1 | I | 25-5/25-8 |
| TDO | Test data out | JTAG | — | 1 | O | 25-5/25-8 |
| TEMP_ANODE | Thermal diode access | Test | — | 2 | I | 25-5/25-8 |
| TEMP_CATHODE | Thermal diode access | Test | — | 2 | I | 25-5/25-8 |
| TEST_SEL | Test select | Test | — | 1 | I | 25-5/25-8 |
| TMS | Test mode select | JTAG | — | 1 | I | 25-5/25-8 |
| TRIG_IN | Watchpoint trigger in | Debug | — | 1 | I | 25-4/25-7 |
| TRIG_OUT | Watchpoint trigger out | Debug | READY | 1 | O | 25-4/25-7 |
| TRST | Test reset | JTAG | — | 1 | I | 25-5/25-8 |
| TSEC_1588_CLK | IEEE 1588 clock | IEEE 1588 | — | 1 | I | 14-2/14-10 |
| TSEC_1588_ CLK_OUT | IEEE 1588 clock out | IEEE 1588 | cfg_ddr_pll2 | 1 | O | 14-2/14-10 |
| TSEC_1588_ PULSE_OUT1 | IEEE 1588 pulse out 1 | IEEE 1588 | cfg_srds2_prtcl2 | 1 | O | 14-2/14-10 |
| TSEC_1588_ PULSE_OUT2 | IEEE 1588 pulse out 2 | IEEE 1588 | cfg_srds2_ref_clk1 | 1 | O | 14-2/14-10 |
| TSEC_1588_ TRIG_IN[0:1] | IEEE 1588 trigger in | IEEE 1588 |  | 2 | I | 14-2/14-10 |
| TSEC_1588_ TRIG_OUT[0:1] | IEEE 1588 alarm out | IEEE 1588 | cfg_ddr_pll[0:1] | 2 | O | 14-2/14-10 |
| TSEC1_COL | TSEC1 collision detect | eTSEC1 | FIFO1_TX_FC | 1 | I | 14-2/14-10 |
| TSEC1_CRS | TSEC1 carrier sense | eTSEC1 | FIFO1_RX_FC | 1 | I/O | 14-2/14-10 |
| TSEC1_GTX_CLK | TSEC1 transmit clock out | eTSEC1 | _ | 1 | O | 14-2/14-10 |
| TSEC1_RX_CLK | TSEC1 receive clock | eTSEC1 | FIFO1_RX_CLK | 1 | I | 14-2/14-10 |
| TSEC1_RX_DV | TSEC1 receive data valid | eTSEC1 | FIFO1_RX_DV | 1 | I | 14-2/14-10 |

**Table 3-2. MPC8536E Alphabetical Signal Reference (continued)**

| Name | Description | Functional Block | Alternate Function(s) | No. of Signals | I/O | Table/ Page |
|---|---|---|---|---|---|---|
| TSEC1_RX_ER | TSEC1 receiver error | eTSEC1 | FIFO1_RX_ER | 1 | I | 14-2/14-10 |
| TSEC1_RXD[7:0] | TSEC1 receive data | eTSEC1 | FIFO1_RXD[7:0] | 8 | I | 14-2/14-10 |
| TSEC1_TX_CLK | TSEC1 transmit clock in | eTSEC1 | FIFO1_TX_CLK | 1 | I | 14-2/14-10 |
| TSEC1_TX_EN | TSEC1 transmit enable | eTSEC1 | FIFO1_TX_EN | 1 | O | 14-2/14-10 |
| TSEC1_TX_ER | TSEC1 transmit error | eTSEC1 | FIFO1_TX_ER/ cfg_tsec1_reduce | 1 | O | 14-2/14-10 |
| TSEC1_TXD[1:0] | TSEC1 transmit data 1 | eTSEC1 | FIFO1_TXD1/ cfg_tsec1_prtcl[1:0] | 2 | O | 14-2/14-10 |
| TSEC1_TXD[7:4] | TSEC1 transmit data 7–4 | eTSEC1 | FIFO1_TXD[7:4]/ cfg_rom_loc[0:3] | 4 | O | 14-2/14-10 |
| TSEC1_TXD2 | TSEC1 transmit data 2 | eTSEC1 | FIFO1_TXD2/ cfg_srds2_prtcl0 | 1 | O | 14-2/14-10 |
| TSEC1_TXD3 | TSEC1 transmit data 3 | eTSEC1 | FIFO1_TXD3/ cfg_eng_use1 | 1 | O | 14-2/14-10 |
| TSEC3_COL | TSEC3 collision detect | eTSEC3 | FIFO3_TX_FC | 1 | I | 14-2/14-10 |
| TSEC3_CRS | TSEC3 carrier sense | eTSEC3 | FIFO3_RX_FC | 1 | I/O | 14-2/14-10 |
| TSEC3_GTX_CLK | TSEC3 transmit clock out | eTSEC3 | — | 1 | O | 14-2/14-10 |
| TSEC3_RX_CLK | TSEC3 receive clock | eTSEC3 | FIFO3_RX_CLK | 1 | I | 14-2/14-10 |
| TSEC3_RX_DV | TSEC3 receive data valid | eTSEC3 | FIFO3_RX_DV | 1 | I | 14-2/14-10 |
| TSEC3_RX_ER | TSEC3 receive error | eTSEC3 | FIFO3_RX_ER | 1 | I | 14-2/14-10 |
| TSEC3_RXD[7:0] | TSEC3 receive data 7–0 | eTSEC3 | FIFO3_RXD[7:0] | 8 | I | 14-2/14-10 |
| TSEC3_TX_CLK | TSEC3 transmit clock in | eTSEC3 | FIFO3_TX_CLK | 1 | I | 14-2/14-10 |
| TSEC3_TX_EN | TSEC3 transmit enable | eTSEC3 | FIFO3_TX_EN | 1 | O | 14-2/14-10 |
| TSEC3_TX_ER | TSEC3 transmit error | eTSEC3 | FIFO3_TX_ER/ cfg_tsec3_reduce | 1 | O | 14-2/14-10 |
| TSEC3_TXD[1:0] | TSEC3 transmit data 1–0 | eTSEC3 | FIFO3_TXD[1:0]/ cfg_tsec3_prtcl[1:0] | 2 | O | 14-2/14-10 |
| TSEC3_TXD[6:4] | TSEC3 transmit data [6:4] | eTSEC3 | FIFO3_TXD[6:4]/ cfg_io_ports[0:2] | 3 | O | 14-2/14-10 |
| TSEC3_TXD2 | TSEC3 transmit data 2 | eTSEC3 | FIFO3_TXD2/ cfg_srds2_prtcl1 | 1 | O | 14-2/14-10 |
| TSEC3_TXD3 | TSEC3 transmit data 3 | eTSEC3 | FIFO3_TXD3/ cfg_srds2_ref_clk0 | 1 | O | 14-2/14-10 |
| TSEC3_TXD7 | TSEC3 transmit data 7 | eTSEC3 | FIFO3_TXD7/ cfg_eng_use2 | 1 | O | 14-2/14-10 |
| $\overline{\text{UART\_CTS}}$[0:1] | DUART clear to send | Dual UART | — | 2 | I | 12-1/12-3 |
| $\overline{\text{UART\_RTS}}$[0:1] | DUART ready to send | Dual UART | — | 2 | O | 12-1/12-3 |
| UART_SIN[0:1] | DUART serial data in | Dual UART | — | 2 | I | 12-1/12-3 |
| UART_SOUT[0:1] | DUART serial data out | Dual UART | — | 2 | O | 12-1/12-3 |

Table 3-2. MPC8536E Alphabetical Signal Reference (continued)

| Name | Description | Functional Block | Alternate Function(s) | No. of Signals | I/O | Table/ Page |
|---|---|---|---|---|---|---|
| $\overline{\text{UDE}}$ | Unconditional debug event | PIC | — | 1 | I | 9-3/9-8 |
| USB1_CLK | USB1 clock | USB1 | — | 1 | I | 21-1/21-3 |
| USB1_D[7:0] | USB1 data | USB1 | — | 8 | I/O | 21-1/21-3 |
| USB1_DIR | USB1 data | USB1 | — | 1 | I | 21-1/21-3 |
| USB1_NXT | USB1 next data | USB1 | — | 1 | I | 21-1/21-3 |
| USB1_PCTL0 | USB1 port control 0 | USB1 | GPIO6 | 1 | O | 21-1/21-3 |
| USB1_PCTL1 | USB1 port control 1 | USB1 | GPIO7 | 1 | O | 21-1/21-3 |
| USB1_PWRFAULT | USB1 power fault | USB1 | — | 1 | I | 21-1/21-3 |
| USB1_STP | USB1 stop | USB1 | cfg_pci_clk | 1 | O | 21-1/21-3 |
| USB2_CLK | USB2 clock | USB2 | — | 1 | I | 21-1/21-3 |
| USB2_D[7:0] | USB2 data | USB2 | — | 8 | I/O | 21-1/21-3 |
| USB2_DIR | USB2 data | USB2 | — | 1 | I | 21-1/21-3 |
| USB2_NXT | USB2 next data | USB2 | — | 1 | I | 21-1/21-3 |
| USB2_PCTL0 | USB2 port control 0 | USB2 | GPIO8 | 1 | O | 21-1/21-3 |
| USB2_PCTL1 | USB2 port control 1 | USB2 | GPIO9 | 1 | O | 21-1/21-3 |
| USB2_PWRFAULT | USB2 power fault | USB2 | — | 1 | I | 21-1/21-3 |
| USB2_STP | USB2 stop | USB2 | cfg_pci_speed | 1 | O | 21-1/21-3 |
| USB3_CLK | USB3 clock | USB3 | — | 1 | I | 21-1/21-3 |
| USB3_D[7:0] | USB3 data | USB3 | — | 8 | I/O | 21-1/21-3 |
| USB3_DIR | USB3 data | USB3 | — | 1 | I | 21-1/21-3 |
| USB3_NXT | USB3 next data | USB3 | — | 1 | I | 21-1/21-3 |
| USB3_STP | USB3 stop | USB3 | — | 1 | O | 21-1/21-3 |

## 3.2 Configuration Signals Sampled at Reset

The signals that serve alternate functions as configuration input signals during system reset are summarized in Table 3-3. The detailed interpretation of their voltage levels during reset is described in Chapter 4, “Reset, Clocking, and Initialization.”

Note that throughout this document, the reset configuration signals are described as being sampled at the negation of $\overline{\text{HRESET}}$. However, there is a setup and hold time for these signals relative to the rising edge of $\overline{\text{HRESET}}$, as described in the *MPC8536E Integrated Processor Hardware Specifications*. Note that the PLL configuration signals have different setup and hold time requirements than the other reset configuration signals.

The reset configuration signals are multiplexed with other functional signals. The values on these signals during reset are interpreted to be logic one or zero, regardless of whether the functional signal name is defined as active-low. Most of the reset configuration signals have internal pull-up resistors so that if the signals are not driven, the default value is high (a one), as shown in the table. Some signals do not have pull-up resistors and must be driven high or low during the reset period. For details about all the signals that require external pull-up resistors, see the *MPC8536E Integrated Processor Hardware Specifications*.

Note that the multiplexing of various signals on the MPC8536E is controlled by the PMUXCR register described in Chapter 23, “Global Utilities.”

**Table 3-3. MPC8536E Reset Configuration Signals**

| Functional Interface | Functional Signal Name | Reset Configuration Name | Default |
|---|---|---|---|
| PCI | $\overline{\text{PCI\_GNT1}}$ | cfg_pci_impd | 1 |
| | $\overline{\text{PCI\_GNT2}}$ | cfg_pci_arb | 1 |
| Ethernet Management | EC_MDC | cfg_eng_use0 | 1 |
| eTSEC1 | TSEC1_TXD[7:4] | cfg_rom_loc[0:3] | 1111 |
| | TSEC1_TXD3 | cfg_eng_use1 | 1 |
| | TSEC1_TXD2 | cfg_srds2_prtcl0 | |
| | TSEC1_TXD[1:0] | cfg_tsec1_prtcl[1:0] | 11 |
| | TSEC1_TX_ER | cfg_tsec1_reduce | |
| eTSEC3 | TSEC3_TXD7 | cfg_eng_use2 | 1 |
| | TSEC3_TXD[6:4] | cfg_io_ports[0:2] | 111 |
| | TSEC3_TXD3 | cfg_srds2_ref_clk0 | |
| | TSEC3_TXD2 | cfg_srds2_prtcl1 | 1 |
| | TSEC3_TXD[1:0] | cfg_tsec3_prtcl[1:0] | 11 |
| | TSEC3_TX_ER | cfg_tsec3_reduce | 1 |
| IEEE 1588 | TSEC_1588_TRIG_OUT[0:1] | cfg_ddr_pll[0:1] | |
| | TSEC_1588_CLK_OUT | cfg_ddr_pll2 | |
| | TSEC_1588_PULSE_OUT1 | cfg_srds2_prtcl2 | |
| | TSEC_1588_PULSE_OUT2 | cfg_srds2_ref_clk1 | |

Table 3-3. MPC8536E Reset Configuration Signals (continued)

| Functional Interface | Functional Signal Name | Reset Configuration Name | Default |
|---|---|---|---|
| eLBC | LAD[0:31] | cfg_gpinput[0:31] | Indeterminate if not driven (no default) |
| | LA27 | cfg_cpu_boot | 1 |
| | LA[28:31] | cfg_sys_pll[0:3] | Must be driven |
| | $\overline{\text{LWE0}}$/$\overline{\text{LBS0}}$/$\overline{\text{LFWE}}$ | cfg_core_speed | 1 |
| | $\overline{\text{LWE}}$[1:3]/$\overline{\text{LBS}}$[1:3] | cfg_host_agt[0:2] | 111 |
| | LBCTL | cfg_core_pll0 | Must be driven |
| | LALE | cfg_core_pll1 | Must be driven |
| | LGPL0/LFCLE | cfg_dram_type | 1 |
| | LGPL1/LFALE | cfg_sys_speed | 1 |
| | LGPL2/$\overline{\text{LOE}}$/$\overline{\text{LFRE}}$ | cfg_core_pll2 | Must be driven |
| | LGPL3/ $\overline{\text{LFWP}}$ | cfg_boot_seq0 | 1 |
| | LGPL5 | cfg_boot_seq1 | 1 |
| Debug | MSRCID0 | cfg_mem_debug | 1 |
| | MSRCID1 | cfg_ddr_debug | 1 |
| USB1 | USB1_STP | cfg_pci_clk | 1 |
| USB2 | USB2_STP | cfg_pci_speed | 1 |

## 3.3 Output Signal States During Reset

When a system reset is recognized ($\overline{\text{HRESET}}$ is asserted), the MPC8536E aborts all current internal and external transactions and releases all bidirectional I/O signals to a high-impedance state. See Chapter 4, "Reset, Clocking, and Initialization," for a complete description of the reset functionality.

During reset, the MPC8536E ignores most input signals (except for reset configuration signals) and drives most output-only signals to an inactive state. Table 3-4 shows the states of the output-only signals (signals that are not multiplexed with other inputs or are not used as reset configuration signals during system reset).

Table 3-4. Output Signal States During System Reset

| Interface | Signal | State During Reset |
|---|---|---|
| DDR Memory | MA[15:0] | High-Z |
| DDR Memory | MPAR_OUT | High-Z |
| DDR Memory | MBA[2:0] | High-Z |
| DDR Memory | $\overline{\text{MCAS}}$ | High-Z |
| DDR Memory | MCK[0:5] | Driven Low |

Table 3-4. Output Signal States During System Reset (continued)

| Interface | Signal | State During Reset |
|---|---|---|
| DDR Memory | $\overline{\text{MCK}}$[0:5] | Driven High |
| DDR Memory | MCKE[0:3] | Driven Low |
| DDR Memory | $\overline{\text{MCS}}$[0:3] | High-Z |
| DDR Memory | MDM[0:8] | High-Z |
| DDR Memory | MODT[0:3] | Driven Low |
| DDR Memory | $\overline{\text{MRAS}}$ | High-Z |
| DDR Memory | $\overline{\text{MWE}}$ | High-Z |
| Dual UART | $\overline{\text{UART\_RTS}}$[0:1] | High-Z |
| eSDHC | SDHC_CLK | High-Z |
| TSEC1 | TSEC1_GTX_CLK | High-Z |
| TSEC1 | TSEC1_TX_EN | High-Z |
| TSEC3 | TSEC3_GTX_CLK | High-Z |
| TSEC3 | TSEC3_TX_EN | High-Z |
| eLBC | LCLK[0:2] | High-Z |
| eLBC | $\overline{\text{LCS}}$[0:4] | High-Z |
| eLBC/DMA | $\overline{\text{LCS6/DMA\_DACK2}}$ | High-Z |
| eLBC/DMA | $\overline{\text{LCS7/DMA\_DDONE2}}$ | High-Z |
| eLBC | LSYNC_OUT | High-Z |
| PCI Express 1 | SD1_TX[3:0], $\overline{\text{SD1\_TX}}$[3:0] | Indeterminate (no default) |
| PCI Express 1/<br>PCI Express 2 | SD1_TX[5:4], $\overline{\text{SD1\_TX}}$[5:4] | Indeterminate (no default) |
| PCI Express 1/<br>PCI Express 2/<br>PCI Express 3 | SD1_TX[7:6], $\overline{\text{SD1\_TX}}$[7:6] | Indeterminate (no default) |
| PIC | $\overline{\text{IRQ\_OUT}}$ | High-Z |
| SATA/SGMII | SD2_TX[1:0], $\overline{\text{SD2\_TX}}$[1:0] | Indeterminate (no default) |
| SPI | SPI_CLK | Driven Low |
| USB3 | USB3_STP | High-Z |
| Clock | CLK_OUT | High-Z |
| Debug | MDVAL | Input—reset config (test only) |
| Debug | MSRCID[2:4] | Input—reset config (test only) |
| Debug | TRIG_OUT/READY/ QUIESCE | Input—reset config (test only) |
| JTAG | TDO | High-Z |
| Power management | ASLEEP | Input—reset config (test only) |

Table 3-4. Output Signal States During System Reset (continued)

| Interface | Signal | State During Reset |
|---|---|---|
| Power management | POWER_EN | Driven High |
| System Control | $\overline{\text{CKSTP\_OUT}}$ | High-Z |
| System Control | $\overline{\text{HRESET\_REQ}}$ | Input—reset config (test only) |

# Chapter 4
# Reset, Clocking, and Initialization

This chapter describes the reset, clocking, and some overall initialization of the MPC8536E, including a definition of the reset configuration signals and the options they select. Additionally, the configuration, control, and status registers are described. Note that other chapters in this book may describe specific aspects of initialization for individual blocks.

## 4.1 Overview

The reset, clocking, and control signals provide many options for device operation. Additionally, many modes are selected with reset configuration signals during a hard reset (assertion of $\overline{\text{HRESET}}$).

## 4.2 External Signal Descriptions

Table 4-1 summarizes the external signals described in this chapter. Table 4-2 and Table 4-3 have detailed signal descriptions, but Table 4-1 contains references to additional sections that contain more information.

**Table 4-1. Signal Summary**

| Signal | I/O | Description | References (Section/Page) |
|---|---|---|---|
| $\overline{\text{HRESET}}$ | I | Hard reset input. Causes a power-on reset (POR) sequence. | 4.4.1.2/4-8 |
| $\overline{\text{HRESET\_REQ}}$ | O | Hard reset request output. An internal block requests that $\overline{\text{HRESET}}$ be asserted. | — |
| $\overline{\text{SRESET}}$ | I | Soft reset input. Causes *mcp* assertion to the core | 4.4.1.1/4-8 |
| READY | O | The MPC8536E has completed the reset operation and is not in a power-down (nap, doze or sleep) or debug state. | 4.4.2/4-9 |
| SYSCLK | I | Primary clock input to the MPC8536E | 4.4.4.1/4-24 |
| DDRCLK | I | Clock input to the MPC8536E that sources the DDR controller complex PLL | 4.4.4.1/4-24 |
| RTC | I | Real time clock input | 4.4.4.4/4-26 |
| SD_REF_CLK/<br>$\overline{\text{SD\_REF\_CLK}}$ | I | SERDES high-speed interface reference clock | 4.4.4.2/4-25 |
| SD2_REF_CLK/<br>$\overline{\text{SD2\_REF\_CLK}}$ | I | Second SERDES high-speed interface reference clock | 4.4.4.2/4-25 |

The following sections describe the reset and clock signals in detail.

### 4.2.1 System Control Signals

Table 4-2 describes some of the system control signals of the MPC8536E. Section 4.4.3, “Power-On Reset Configuration,” describes the signals that also function as reset configuration signals. Note that the $\overline{\text{CKSTP\_IN}}$ and $\overline{\text{CKSTP\_OUT}}$ signals are described in Chapter 23, “Global Utilities.”

**Table 4-2. System Control Signals—Detailed Signal Descriptions**

<table>
<tr><th>Signal</th><th>I/O</th><th colspan="2">Description</th></tr>
<tr><td rowspan="3">$\overline{\text{HRESET}}$</td><td rowspan="3">I</td><td colspan="2">Hard reset. Causes the MPC8536E to abort all current internal and external transactions and set all registers to their default values. $\overline{\text{HRESET}}$ may be asserted completely asynchronously with respect to all other signals.</td></tr>
<tr><td><b>State Meaning</b></td><td>Asserted/Negated—See Chapter 3, “Signal Descriptions,” and Section 4.4.3, “Power-On Reset Configuration,” for more information on the interpretation of device signals during reset.</td></tr>
<tr><td><b>Timing</b></td><td>Assertion/Negation—The <i>MPC8536E Integrated Processor Hardware Specifications</i> gives specific timing information for this signal and the reset configuration signals.</td></tr>
<tr><td rowspan="3">$\overline{\text{HRESET\_REQ}}$</td><td rowspan="3">O</td><td colspan="2">Hard reset request. Indicates to the board (system in which the MPC8536E is embedded) that a condition requiring the assertion of HRESET has been detected.</td></tr>
<tr><td><b>State Meaning</b></td><td>Asserted—A watchdog timer or a boot sequencer failure (see Section 11.4.5, “Boot Sequencer Mode”) has triggered a request for hard reset.<br>Negated—Indicates no reset request.</td></tr>
<tr><td><b>Timing</b></td><td>Assertion/Negation—May occur any time, synchronous to the core complex bus clock. Once asserted, $\overline{\text{HRESET\_REQ}}$ does not negate until $\overline{\text{HRESET}}$ is asserted.</td></tr>
<tr><td rowspan="3">$\overline{\text{SRESET}}$</td><td rowspan="3">I</td><td colspan="2">Soft reset. Causes a machine check interrupt to the e500 core. Note that if the e500 core is not configured to process machine check interrupts, the assertion of $\overline{\text{SRESET}}$ causes a core checkstop. $\overline{\text{SRESET}}$ need not be asserted during a hard reset.</td></tr>
<tr><td><b>State Meaning</b></td><td>Asserted—Asserting $\overline{\text{SRESET}}$ causes a machine check interrupt (edge sensitive) to the e500 core. $\overline{\text{SRESET}}$ has no effect while $\overline{\text{HRESET}}$ is asserted. However, the POR sequence is paused if $\overline{\text{SRESET}}$ is asserted during POR.</td></tr>
<tr><td><b>Timing</b></td><td>Assertion—May occur at any time, asynchronous to any clock.<br>Negation—Must be asserted for at least two <i>CCB_clk</i> cycles.</td></tr>
<tr><td rowspan="3">READY</td><td rowspan="3">O</td><td colspan="2">Ready. Multiplexed with TRIG_OUT and $\overline{\text{QUIESCE}}$. See Chapter 25, “Debug Features and Watchpoint Facility,” for more information on TOSR and TRIG_OUT.</td></tr>
<tr><td><b>State Meaning</b></td><td>Asserted—Indicates that the MPC8536E has completed the reset operation and is not in a power-down state (nap, doze, or sleep) when TOSR[SEL] equals 0b000. See Section 4.4.2, “Power-On Reset Sequence,” for more information.</td></tr>
<tr><td><b>Timing</b></td><td>Assertion/Negation—Initial assertion of READY after reset is synchronous with SYSCLK. Subsequent assertion/negation due to power down modes occurs asynchronously.</td></tr>
</table>

### 4.2.2 Clock Signals

Table 4-3 describes the overall clock signals. Note that some clock signals are specific to blocks within the device, and although some of their functionality is described in Section 4.4.4, "Clocking," they are defined in detail in their respective chapters.

Note that there is also a CLK_OUT signal; the signal driven on the CLK_OUT pin is selectable and described in Section 23.4.1.25, "Clock Out Control Register (CLKOCR)."

**Table 4-3. Clock Signals—Detailed Signal Descriptions**

| Signal | I/O | Description |
|---|---|---|
| SYSCLK | I | System clock/PCI clock (SYSCLK/PCI_CLK). SYSCLK is the primary clock input to the MPC8536E. It is the clock source for the e500 core and for all devices and interfaces that operate synchronously with the core. It is multiplied up with a phased-lock loop (PLL) to create the core complex bus (CCB) clock (also called the platform clock), which is used by virtually all of the synchronous system logic, including the L2 cache, the DDR SDRAM and local bus memory controllers, and other internal blocks such as the DMA and interrupt controllers. The CCB clock, in turn, feeds the PLL in the e500 core and the PLL that creates the local bus memory clocks.<br>When the PCI interface is used, SYSCLK also functions as the PCI_CLK signal. Note that this is true whether the device is in agent or host mode. The MPC8536E does not provide a separate PCI_CLK output in host mode. |
| | | **Timing** Assertion/Negation—See the *MPC8536E Integrated Processor Hardware Specifications* for specific timing information for this signal. |
| DDRCLK | I | DDR controller complex clock (DDRCLK). DDRCLK is the clock source for the DDR memory controller complex except in the case where synchronous mode of operation is selected (see Section 4.4.3.3, "DDR PLL Ratio Configuration"). This clock input is multipled up with a phased-lock loop (PLL) to create the DDR controller complex clock. The DDR memory controller complex clock is the DDR data rate on the external interface unless the given controller is configured to run in half speed (see Section 23.4.1.24, "DDR Clock Disable Register (DDRCLKDR)"). |
| | | **Timing** Assertion/Negation—See the *MPC8536E Integrated Processor Hardware Specifications* for specific timing information for this signal. |
| RTC | I | Real time clock. May be used (optionally) to clock the time base of the e500 core. The RTC timing specifications are given in the *MPC8536E Integrated Processor Hardware Specifications*, but the maximum frequency should be less than one-quarter of the CCB frequency. See Section 4.4.4.4, "Real Time Clock." This signal can also be used (optionally) to clock the global timers in the programmable interrupt controller (PIC). |
| | | **Timing** Assertion/Negation—See the *MPC8536E Integrated Processor Hardware Specifications* for specific timing information for this signal. |

## 4.3 Memory Map/Register Definition

This section describes the configuration and control registers that control access to the configuration space and to the boot code as well as guidelines for accessing these regions. It also contains a brief description of the boot sequencer which may be used to initialize configuration registers or memory before the CPU is released to boot.

### 4.3.1 Local Configuration Control

Table 4-4 shows the memory map for local configuration control registers.

Table 4-4. Local Configuration Control Register Map

| Local Memory Offset (Hex) | Register | Access | Reset | Section/Page |
|---|---|---|---|---|
| 0x0_0000 | CCSRBAR—Configuration, control, and status registers base address register | R/W | 0x000F_F700 | 4.3.1.1.2/4-5 |
| 0x0_0008 | ALTCBAR—Alternate configuration base address register | R/W | 0x0000_0000 | 4.3.1.2.1/4-6 |
| 0x0_0010 | ALTCAR—Alternate configuration attribute register | R/W | 0x0000_0000 | 4.3.1.2.1/4-6 |
| 0x0_0020 | BPTR—Boot page translation register | R/W | 0x0000_0000 | 4.3.1.3.1/4-7 |

#### 4.3.1.1 Accessing Configuration, Control, and Status Registers

The configuration, control, and status registers are memory mapped. The set of configuration, control, and status registers occupies a 1-Mbyte region of memory. Their location is programmable using the CCSR base address register (CCSRBAR). The default base address for the configuration, control, and status registers is 0xFF70_0000 (CCSRBAR = 0x000F_F700). CCSRBAR itself is part of the local access block of CCSR memory, which begins at offset 0x0 from CCSRBAR. Because CCSRBAR is at offset 0x0 from the beginning of the local access registers, CCSRBAR always points to itself. The contents of CCSRBAR are broadcast internally in the MPC8536E to all functional units that need to be able to identify or create configuration transactions.

##### 4.3.1.1.1 Updating CCSRBAR

Updates to CCSRBAR that relocate the entire 1-Mbyte region of configuration, control, and status registers require special treatment. The effect of the update must be guaranteed to be visible by the mapping logic before an access to the new location is seen. To make sure this happens, these guidelines should be followed:

- CCSRBAR should be updated during initial configuration of the device when only one host or controller has access to the device.
  - If the boot sequencer is being used to initialize, it is recommended that the boot sequencer set CCSRBAR to its desired final location.
  - If an external host on PCI is configuring the device, it should set CCSRBAR to the desired final location before the e500 core is released to boot.
  - If the e500 core is initializing the device, it should set CCSRBAR to the desired final location before enabling other I/O devices to access the device.
- When the e500 core is writing to CCSRBAR, it should use the following sequence:
  - Read the current value of CCSRBAR using a load word instruction followed by an **isync**. This forces all accesses to configuration space to complete.
  - Write the new value to CCSRBAR.

- Perform a load of an address that does not access configuration space or the on-chip SRAM, but has an address mapping already in effect (for example, boot ROM). Follow this load with an **isync**.
- Read the contents of CCSRBAR from its new location, followed by another **isync**.

#### 4.3.1.1.2 Configuration, Control, and Status Registers Base Address Register (CCSRBAR)

Figure 4-1 shows the fields of CCSRBAR.

Offset 0x0_0000 Access: Read/Write

| | 0 … 7 | 8 … 23 | 24 … 31 |
|---|---|---|---|
| R<br>W | — | BASE_ADDR | — |
| Reset | 0 0 0 0 0 0 0 0 | 0 0 0 0 1 1 1 1 1 1 1 1 0 1 1 1 | 0 0 0 0 0 0 0 0 |

**Figure 4-1. Configuration, Control, and Status Registers Base Address Register (CCSRBAR)**

Table 4-5 defines the bit fields of CCSRBAR.

**Table 4-5. CCSRBAR Bit Settings**

| Bits | Name | Description |
|---|---|---|
| 0–7 | — | Write reserved, read = 0. |
| 8–23 | BASE_ADDR | Identifies the16 most-significant address bits of the 36-bit window used for configuration accesses. The base address is aligned on a 1-Mbyte boundary. |
| 24–31 | — | Write reserved, read = 0 |

### 4.3.1.2 Accessing Alternate Configuration Space

An alternate configuration space can be accessed by configuring the ALTCBAR and ALTCAR registers. These are intended to be used with the boot sequencer to allow the boot sequencer to access an alternate 1-Mbyte region of configuration space. By loading the proper boot sequencer command in the serial ROM, the base address in the ALTCBAR can be combined with the 20 bits of address offset supplied from the serial ROM to generate a 36-bit address that is mapped to the target specified in ALTCAR. Thus, by configuring these registers, the boot sequencer has access to the entire memory map, one 1-Mbyte block at a time. See Section 11.4.5, “Boot Sequencer Mode,” for more information.

**NOTE**

> The enable bit in the ALTCAR register should be cleared either by the boot sequencer or by the boot code that executes after the boot sequencer has completed its configuration operations. This prevents problems with incorrect mappings if subsequent configuration of the local access windows uses a different target mapping for the address specified in ALTCBAR.

#### 4.3.1.2.1 Alternate Configuration Base Address Register (ALTCBAR)

Figure 4-2 shows the fields of ALTCBAR.

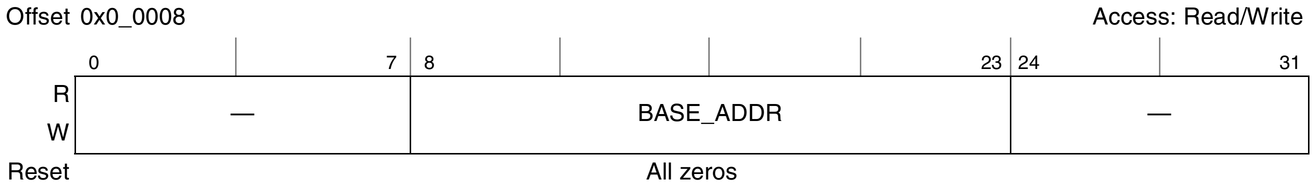


**Figure 4-2. Alternate Configuration Base Address Register (ALTCBAR)**

Table 4-6 defines the bit fields of ALTCBAR.

**Table 4-6. ALTCBAR Bit Settings**

| Bits | Name | Description |
|---|---|---|
| 0–7 | — | Write reserved, read = 0 |
| 8–23 | BASE_ADDR | Identifies the16 most significant address bits of an alternate window used for configuration accesses. Like CCSRBAR, this alternate window has a fixed size of 1 Mbyte. |
| 24–31 | — | Write reserved, read = 0 |

#### 4.3.1.2.2 Alternate Configuration Attribute Register (ALTCAR)

Figure 4-3 shows the fields of ALTCAR.

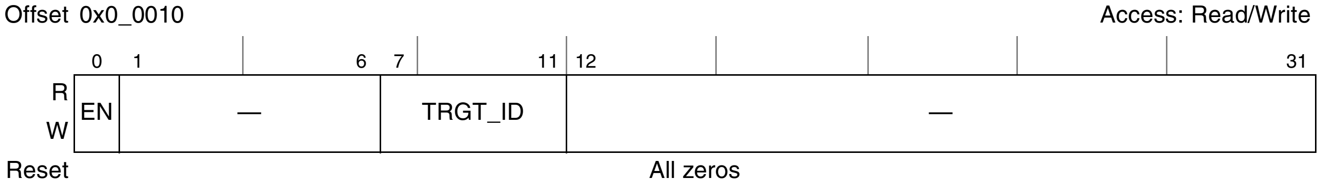


**Figure 4-3. Alternate Configuration Attribute Register (ALTCAR)**

Table 4-7 defines ALTCAR fields.

**Table 4-7. ALTCAR Bit Settings**

| Bits | Name | Description |
|---|---|---|
| 0 | EN | Enable second configuration window. Like CCSRBAR, it has a fixed size of 1 Mbyte.<br>0 Second configuration window is disabled.<br>1 Second configuration window is enabled. |
| 1–6 | — | Write reserved, read = 0 |

Table 4-7. ALTCAR Bit Settings (continued)

| Bits | Name | Description |
|---|---|---|
| 7–11 | TRGT_ID | Identifies the device ID to target when a transaction hits in the 1-Mbyte address range defined by the second configuration window.<br>00000 PCI Interface<br>00001 PCI Express 2<br>00010 PCI Express 1<br>00011 PCI Express 3<br>00100 Local bus controller<br>00101–01011 Reserved<br>01000 Configuration, control, status registers<br>01001–01110 Reserved<br>01111 Local memory—DDR SDRAM and on-chip SRAM |
| 12–31 | — | Write reserved, read = 0 |

### 4.3.1.3 Boot Page Translation

When the e500 core comes out of reset, its MMU has one 4-Kbyte page defined at 0x0_FFFF_F*nnn*. The core begins execution with the instruction at effective address 0x0_FFFF_FFFC. To get this instruction, the core's first instruction fetch is a burst read of boot code from effective address 0x0_FFFF_FFE0. For systems in which the boot code resides at a different address, the MPC8536E provides boot page translation capability. Boot page translation is controlled by the boot page translation register (BPTR).

The boot sequencer can enable boot page translation, or the boot page translation can be set up by an external host when the device is configured to be in boot holdoff mode. If translation is to be performed to a page outside the default boot ROM address range defined in the MPC8536E (8 Mbytes at 0x0_FF80_0000 to 0x0_FFFF_FFFF as defined in Section 4.4.3.6, "Boot ROM Location"), the external host or boot sequencer must then also set up a local access window to define the routing of the boot code fetch to the target interface that contains the boot code, because the BPTR defines only the address translation, not the target interface. See Section 2.1, "Local Memory Map Overview and Example," and Section 11.4.5, "Boot Sequencer Mode," for more information.

#### 4.3.1.3.1 Boot Page Translation Register (BPTR)

Figure 4-4 shows the fields of BPTR.

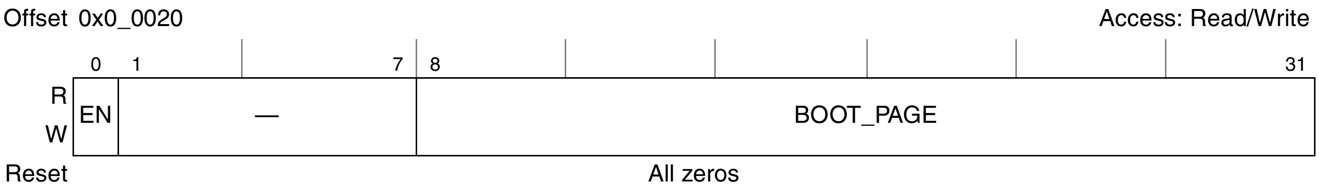


Figure 4-4. Boot Page Translation Register (BPTR)

Table 4-8 describes BPTR bit settings.

Table 4-8. BPTR Bit Settings

| Bits | Name | Description |
|---|---|---|
| 0 | EN | Boot page translation enable<br>0 Boot page is not translated.<br>1 Boot page is translated as defined in the BPTR[BOOT_PAGE] parameter. |
| 1–7 | — | Write reserved, read = 0 |
| 8–31 | BOOT_PAGE | Translation for boot page. If enabled, the high order 24 bits of accesses to 0x0_FFFF_F*nnn* are replaced with this value. |

### 4.3.2 Boot Sequencer

The boot sequencer is a DMA engine that accesses a serial ROM on the I$^2$C interface and writes data to CCSR memory or the memory space pointed to by the alternate configuration base address register (ALTCBAR). See Section 4.3.1.2, “Accessing Alternate Configuration Space.” The boot sequencer is enabled by reset configuration pins as described in Section 4.4.3.11, “Boot Sequencer Configuration.” If the boot sequencer is enabled, the e500 core is held in reset until the boot sequencer has completed its operation. For more details, see Section 11.4.5, “Boot Sequencer Mode,” in the I$^2$C chapter.

## 4.4 Functional Description

This section describes the various ways to reset the MPC8536E, the POR configurations, and the clocking on the device.

### 4.4.1 Reset Operations

The MPC8536E has reset input signals for hard and soft reset operation.

#### 4.4.1.1 Soft Reset

Assertion of $\overline{\text{SRESET}}$ causes a machine check interrupt to the e500 core. When this occurs, the soft reset flag is recorded in the machine check summary register (MCPSUMR) in the global utilities block so that software can identify the machine check as a soft reset condition. See the *PowerPC e500 Core Complex Reference Manual* for more information on the machine check interrupt and Section 23.4.1.16, “Machine Check Summary Register (MCPSUMR),” for more information on the setting of the soft reset flag. Note that if $\overline{\text{SRESET}}$ is asserted before the e500 core is configured to handle a machine check interrupt, a core checkstop condition occurs, which causes $\overline{\text{CKSTP\_OUT}}$ to assert.

#### 4.4.1.2 Hard Reset

The device can be completely reset by the assertion of the $\overline{\text{HRESET}}$ input. The assertion of this signal by external logic is the equivalent of a POR and causes the sequence of events described in Section 4.4.2, “Power-On Reset Sequence.”

Refer to the *MPC8536E Integrated Processor Hardware Specifications* for the timing requirements for $\overline{\text{HRESET}}$ assertion and negation.

The hard reset request output signal ($\overline{\text{HRESET\_REQ}}$) indicates to external logic that a hard reset is being requested by hardware or software. Hardware causes this signal to assert for a boot sequencer failure (see Section 11.4.5, “Boot Sequencer Mode,” and Section 11.4.5.2, “EEPROM Data Format”) or when the e500 watchdog timer is configured to cause a reset request when it expires. Software may request a hard reset by setting a bit in a global utilities register; see Section 23.4.1.22, “Reset Control Register (RSTCR).”

## 4.4.2 Power-On Reset Sequence

The POR sequence for the MPC8536E is as follows:

1. Power is applied to meet the specifications in the *MPC8536E Integrated Processor Hardware Specifications*.
2. The system asserts $\overline{\text{HRESET}}$ and $\overline{\text{TRST}}$, causing all registers to be initialized to their default states and most I/O drivers to be three-stated (some clock, clock enabled, and system control signals are active).
3. The system applies a stable SYSCLK signal and stable PLL configuration inputs, and the device PLL begins locking to SYSCLK.
4. System negates $\overline{\text{HRESET}}$ after its required hold time and after POR configuration inputs have been valid for at least 4 SYSCLK cycles.

**NOTE:**

> If the JTAG signals are not used, then $\overline{\text{TRST}}$ may be tied negated. It is recommended that $\overline{\text{TRST}}$ not remain asserted after the negation of $\overline{\text{HRESET}}$. $\overline{\text{TRST}}$ may be connected directly to $\overline{\text{HRESET}}$.
>
> There is no need to assert the $\overline{\text{SRESET}}$ signal when $\overline{\text{HRESET}}$ is asserted. If $\overline{\text{SRESET}}$ remains asserted upon negation of $\overline{\text{HRESET}}$, the POR sequence will be paused after the e500 core PLL is locked and before the e500 reset is negated. The POR sequence will be resumed when $\overline{\text{SRESET}}$ is negated.

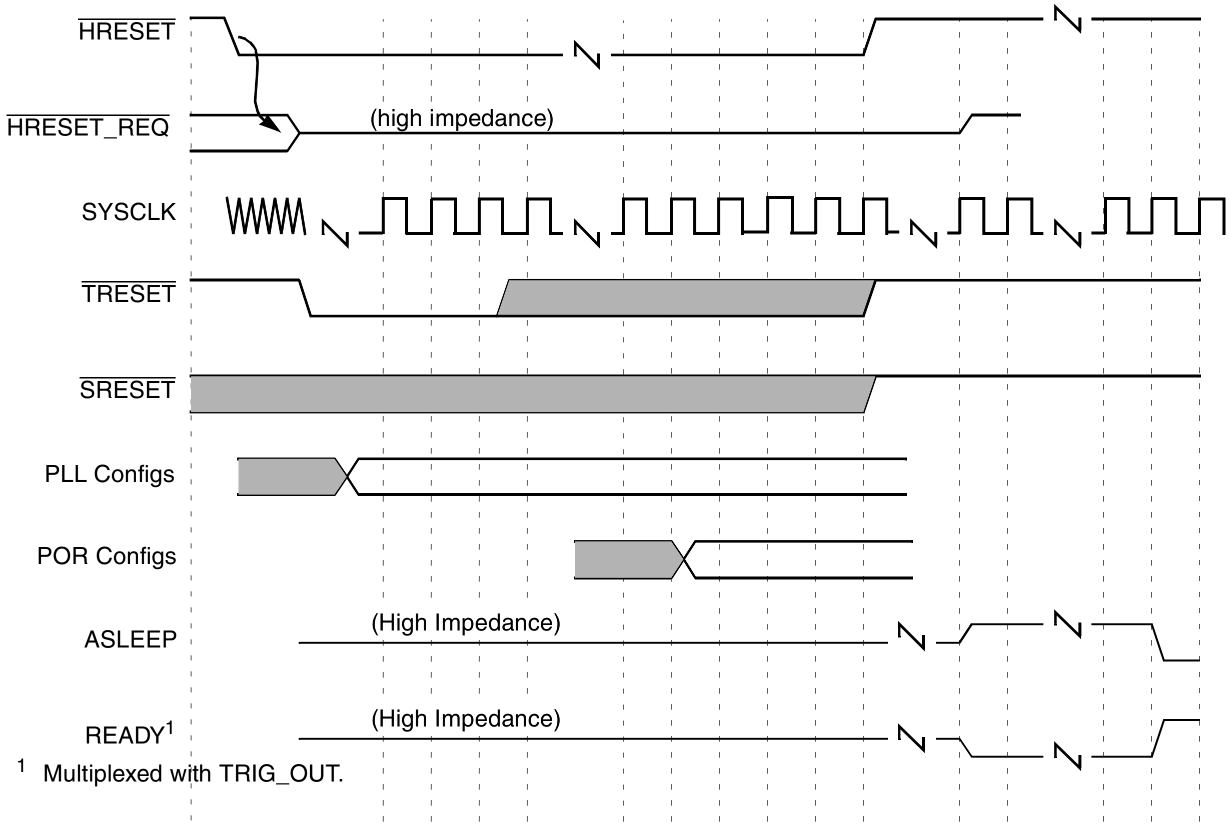
5. MPC8536E enables I/O drivers.
6. The MPC8536E PCI interface can assert $\overline{\text{DEVSEL}}$ in response to configuration cycles.
7. The e500 PLL configuration inputs are applied, allowing the e500 PLL to begin locking to the device clock (the CCB clock).
8. The CCB clock is cycled for approximately 50 µs to lock the e500 PLL.
9. The internal hard reset to the e500 core is negated and soft resets are negated to the PLLs and other remaining I/O blocks. The PLLs begin to lock.
10. When PLL locking is completed, the local bus FCM is released provided NAND Flash is configured as the boot device, as described in Section 4.4.3.6, “Boot ROM Location.” Once the FCM finishes loading the pages from the NAND Flash device, the boot sequencer, if enabled, is allowed to progress, causing it to load configuration data from serial ROMs on the I2C1 interface, as described in Section 11.4.5, “Boot Sequencer Mode.”

11. When the local bus FCM and boot sequencer complete, the PCI Express interfaces begin training, the PCI and PCI Express interfaces are released to accept external requests, and the boot vector fetched by the e500 core is allowed to proceed unless processor booting is further held off by POR configuration inputs as described in Section 4.4.3.10, "CPU Boot Configuration." The MPC8536E is now in its ready state.
12. The ASLEEP signal negates synchronized to a rising edge of SYSCLK, indicating the ready state. The ready state is also indicated by the assertion of READY/TRIG_OUT if TOSR[SEL] = 000. In this case, READY is asserted with the same rising edge of SYSCLK, to indicate that the device has reached its ready state. See Section 25.3.4.1, "Trigger Out Source Register (TOSR)," for more information on this register.

    Asserting READY allows external system monitors to know basic device status, for example, exactly when it emerges from reset, or if the device is in a low-power mode. For more information on the debug functions of TRIG_OUT, see Section 25.3.4, "Trigger Out Function." For more information about power management states, see Section 23.4.1, "Register Descriptions."

More information on system booting is given later in this chapter. See Section 4.5.1, "System Boot."

Figure 4-5 shows a timing diagram of the POR sequence.



[1] Multiplexed with TRIG_OUT.

**Figure 4-5. Power-On Reset Sequence**

## 4.4.3 Power-On Reset Configuration

Various device functions are initialized by sampling certain signals during the assertion of $\overline{\text{HRESET}}$. The values of all these signals are sampled into registers while $\overline{\text{HRESET}}$ is asserted. These inputs are to be pulled high or low by external resistors. During $\overline{\text{HRESET}}$, all other signal drivers connected to these signals must be in the high-impedance state.

Most POR configuration signals have internal pull-up resistors so that if the desired setting is high, there is no need for a pull-up resistor on the board. Other POR configuration signals do not use pull-ups and therefore must be pulled high or low. Refer to the *MPC8536E Integrated Processor Hardware Specifications* for proper resistor values to be used for pulling POR configuration signals high or low.

This section describes the functions and modes configured by POR configuration signals. Note that many reset configuration settings are accessible to software through the following read-only memory-mapped registers described in Chapter 23, “Global Utilities”:

- POR PLL status register (PORPLLSR)
- POR boot mode status register (PORBMSR)
- POR I/O impedance status and control register (PORIMPSCR)
- POR device status register (PORDEVSR)
- POR debug mode status register (PORDBGMSR)
- General-purpose POR configuration register (GPPORCR)—Reports the value on LAD[0:31] during POR (can be used to external system configuration)

**NOTE**

In the following tables, the binary value 0b0 represents a signal pulled down to GND and a value of 0b1 represents a signal pulled up to $V_{DD}$, regardless of the sense of the functional signal name on the signal.

### 4.4.3.1 System PLL Ratio

The system PLL inputs, shown in Table 4-9, establish the clock ratio between the SYSCLK input and the platform clock used by the MPC8536E. The platform clock, also called the CCB clock, drives the L2 cache, the DDR SDRAM data rate, and the e500 core complex bus (CCB). See Section 4.4.4.2.1, “Minimum Frequency Requirements,” for optimal selection of this ratio with regard to available high-speed interface widths and frequencies. Note that the values latched on these signals during POR are accessible in the PORPLLSR (POR PLL status register), as described in Section 23.4.1.1, “POR PLL Status Register (PORPLLSR).”

Note that x8 PCI Express is only available at CCB clock rates of 527 MHz and above.

Table 4-9. CCB Clock PLL Ratio

| Functional Signals | Reset Configuration Name | Value (Binary) | CCB Clock : SYSCLK Ratio |
|---|---|---|---|
| LA[28:31]<br>Default (1111) | cfg_sys_pll[0:3] | 0000 | 16 : 1 |
| | | 0001 | Reserved |
| | | 0010 | Reserved |
| | | 0011 | 3 : 1 |
| | | 0100 | 4 : 1 |
| | | 0101 | 5 : 1 |
| | | 0110 | 6 : 1 |
| | | 0111 | Reserved |
| | | 1000 | 8 : 1 |
| | | 1001 | 9 : 1 |
| | | 1010 | 10 : 1 |
| | | 1011 | Reserved |
| | | 1100 | 12 : 1 |
| | | 1101 | Reserved |
| | | 1110 | Reserved |
| | | 1111 | Reserved (default) |

### 4.4.3.2 e500 Core PLL Ratio

Table 4-10 describes the e500 core clock PLL inputs that program the core PLL and establish the ratio between the e500 core clock and the e500 core complex bus (CCB) clock. Note that the values latched on these signals during POR are accessible through the memory-mapped PORPLLSR, as described in Section 23.4.1.1, “POR PLL Status Register (PORPLLSR),” and also in the e500 core HID1 register, as described in the *PowerPC e500 Core Family Reference Manual* and in Section 5.3, “Summary of Core Integratation Details.”

Table 4-10. e500 Core Clock PLL Ratios

| Functional Signals | Reset Configuration Name | Value (Binary) | e500 Core: CCB ClockRatio |
|---|---|---|---|
| LBCTL, LALE, LGPL2/$\overline{\text{LOE}}$/$\overline{\text{LFRE}}$<br>Default (111) | cfg_core_pll[0:2] | 000 | 4 : 1 |
| | | 001 | 9 : 2 (4.5:1) |
| | | 010 | Reserved |
| | | 011 | 3 : 2 (1.5 : 1) |
| | | 100 | 2 : 1 |
| | | 101 | 5 : 2 (2.5:1) |
| | | 110 | 3 : 1 |
| | | 111 | 7 : 2 (3.5 : 1) (default) |

### 4.4.3.3 DDR PLL Ratio Configuration

The DDR PLL inputs, shown in Table 4-11, establish the clock ratio between the DDRCLK input and the DDR complex clock. The DDR complex clock drives the DDR data rate, which is 2 times the rate at which commands are issued on the DDR interface.

This DDR complex clock domain is asynchronous to the platform clock or CCB clock domain, and is sourced from a separate PLL than the rest of the platform, unless the DDR PLL encoding for synchronous mode operation is selected. When synchronous mode is selected, the DDR complex is driven by the CCB clock, which becomes the DDR data rate.

There is no default value for this PLL ratio; these signals must be pulled to the desired values. Note that the encoded values latched on these signals during POR—and not the actual values on the pins—are accessible in the PORPLLSR (POR PLL status register), as described in Section 23.4.1.1, “POR PLL Status Register (PORPLLSR).”

**Table 4-11. DDR Complex Clock PLL Ratios**

| Functional Signals | Reset Configuration Name | Value (Binary) | DDR Complex Clock: DDR Clock Ratio |
|---|---|---|---|
| TSEC_1588_TRIG_OUT[0:1], TSEC_1588_CLK_OUT<br>No Default | cfg_ddr_pll[0:2] | 000 | 3:1 |
| | | 001 | 4:1 |
| | | 010 | 6:1 |
| | | 011 | 8:1 |
| | | 100 | 10:1 |
| | | 101 | 12:1 |
| | | 110 | Reserved |
| | | 111 | Synchronous mode |

### 4.4.3.4 System Speed Configuration

The SYSCLK speed configuration inputs, shown in Table 4-12 configure internal logic for proper operation with the SYSCLK clock frequencies in use. The default setting is appropriate for SYSCLK operating above 66 MHz; for low speed operation (SYSCLK at or below 66 MHz) this POR configuration input should be low during $\overline{\text{HRESET}}$. If this configuration is not set properly, behavior of the system may be unreliable. Note that the value latched on this signal during POR is accessible through the memory-mapped PORDEVSR, described inSection 23.4.1.4, “POR Device Status Register (PORDEVSR).”

**Table 4-12. System Speed Configuration**

| Functional Signal | Reset Configuration Name | Value (Binary) | Meaning |
|---|---|---|---|
| LGPL1/LFALE<br>Default (1) | cfg_sys_speed | 0 | SYSCLK frequency at or below 66 MHz |
| | | 1 | SYSCLK frequency above 66 MHz |

### 4.4.3.5 Core Speed Configuration

The core speed configuration inputs, shown in Table 4-13 configure internal logic for proper operation with the core clock frequencies in use. The default setting is appropriate for the core operating above 800 MHz; for low-speed operation (core at or below 800 MHz) this POR configuration input should be low during $\overline{\text{HRESET}}$. If this configuration is not set properly, behavior of the system may be unreliable. Note that the value latched on this signal during POR is accessible through the memory-mapped PORDEVSR, described in Section 23.4.1.4, “POR Device Status Register (PORDEVSR).”

**Table 4-13. Core Speed Configuration**

| Functional Signal | Reset Configuration Name | Value (Binary) | Meaning |
|---|---|---|---|
| $\overline{\text{LWE0}}$/$\overline{\text{LBS0}}$/$\overline{\text{LFWE}}$ | cfg_core_speed | 0 | Core frequency at or below 800 MHz |
| Default (1) | | 1 | Core frequency above 800 MHz |

### 4.4.3.6 Boot ROM Location

The device defines the default boot ROM address range to be 8 Mbytes at address 0x0_FF80_0000 to 0x0_FFFF_FFFF. However, which peripheral interface handles these boot ROM accesses can be selected at power on.

The boot ROM location inputs, shown in Table 4-14, select the physical location of boot ROM. Accesses to the boot vector and the default boot ROM region of the local address map are directed to the interface specified by these inputs.

Boot from eSPI or SD/MMC is supported by the MPC8536E using an on-chip ROM which contains the basic eSPI or eSDHC device driver and the code to perform block copy from eSPI EEPROM or SD/MMC card to DDR memory. Selecting on-chip ROM in boot ROM location will cause the e500 CPU to fetch data from the on-chip ROM.

Table 4-14. Boot ROM Location

| Functional Signals | Reset Configuration Name | Value (Binary) | Meaning |
|---|---|---|---|
| TSEC1_TXD[7:4]<br>Default (1111) | cfg_rom_loc[0:3] | 0000 | PCI |
| | | 0001 | PCI Express 1 |
| | | 0010 | PCI Express 2 |
| | | 0011 | PCI Express 3 |
| | | 0100 | DDR controller |
| | | 0101 | Reserved |
| | | 0110 | On-chip boot ROM eSPI configuration |
| | | 0111 | On-chip boot ROM eSDHC configuration |
| | | 1000 | Local bus FCM—8-bit NAND Flash small page ECC enabled |
| | | 1001 | Local bus FCM—8-bit NAND Flash small page ECC disabled |
| | | 1010 | Local bus FCM—8-bit NAND Flash large page ECC enabled |
| | | 1011 | Local bus FCM—8-bit NAND Flash large page ECC disabled |
| | | 1100 | Reserved |
| | | 1101 | Local bus GPCM—8-bit ROM |
| | | 1110 | Local bus GPCM—16-bit ROM |
| | | 1111 | Local bus GPCM—32-bit ROM (default) |

Note that the values latched on these signals during POR are accessible through the memory-mapped PORBMSR (POR boot mode status register) described in Section 23.4.1.2, “POR Boot Mode Status Register (PORBMSR).”

See Section 2.1, “Local Memory Map Overview and Example,” for an example memory map that relies on the default boot ROM values. Also, see Section 4.3.1.3.1, “Boot Page Translation Register (BPTR),” for information on translation of the boot page.

#### 4.4.3.7 Host/Agent Configuration

The host/agent reset configuration inputs, shown in Table 4-15, configure the MPC8536E to act as a host or as an agent of a master on another interface. In host mode, the device is immediately enabled to master transactions to the PCI interface. If the device is an agent on the PCI or PCI Express interfaces, then the device is disabled from mastering transactions on that interface until the external host enables it to do so. The external host does this by setting the control registers of the MPC8536E’s interfaces appropriately. See details in the PCI and PCI Express, programming models described in Chapter 16, “PCI Bus Interface,” and Chapter 17, “PCI Express Interface Controller,” respectively.

Note that the values latched on these signals during POR are accessible through the memory-mapped PORBMSR (POR boot mode status register) described in Section 23.4.1.2, “POR Boot Mode Status Register (PORBMSR).”

**Table 4-15. Host/Agent Configuration**

| Functional Signals | Reset Configuration Name | Value (Binary) | Meaning |
|---|---|---|---|
| $\overline{\text{LWE}}$[1:3]/$\overline{\text{LBS}}$[1:3]<br>Default (111) | cfg_host_agt[0:2] | 000 | Reserved |
| | | 001 | MPC8536E acts as an endpoint on PCI Express 3 interface. It acts as the host/root complex for all other PCI/PCI Express interfaces. |
| | | 010 | Reserved |
| | | 011 | MPC8536E acts as an endpoint on PCI Express 2 interface. It acts as the host/root complex for all other PCI/PCI Express interfaces. |
| | | 100 | Reserved |
| | | 101 | MPC8536E acts as an endpoint on PCI Express 1 interface. It acts as the host/root complex for all other PCI/PCI Express interfaces. |
| | | 110 | MPC8536E acts as an agent of an external host on its PCI interface. It acts as a root complex for all PCI Express interfaces. |
| | | 111 | MPC8536E acts as the host processor/root complex on all interfaces (default). |

### 4.4.3.8 SerDes1 I/O Port Selection

The device can be configured with different I/O ports active on SerDes1. Table 4-16 shows the configuration of I/O ports and bit rates (and required reference clocks) that are possible for SerDes1 interfaces.

**Table 4-16. SerDes1 I/O Port Selection**

| Functional Signal | Reset Configuration Name | Value (Binary) | Meaning |
|---|---|---|---|
| TSEC3_TXD[6:4]<br>Default (111) | cfg_io_ports[0:2] | 000 | Reserved |
| | | 001 | All 3 PCI Express ports powered down |
| | | 010 | PCI Express 1 (x4) (2.5Gbps) → SerDes1 Lanes A–D<br>SerDes1 Lanes E–H powered down |
| | | 011 | PCI Express 1 (x8) (2.5Gbps) → SerDes1 Lanes A–H[1] |
| | | 100 | Reserved |
| | | 101 | PCI Express 1 (x4) (2.5 Gbps) → SerDes1 Lanes A–D<br>PCI Express 2 (x4) (2.5 Gbps) → SerDes1 Lanes E–H |
| | | 110 | Reserved |
| | | 111 | PCI Express 1 (x4) (2.5 Gbps) → SerDes1 Lanes A–D<br>PCI Express 2 (x2) (2.5 Gbps) → SerDes1 Lanes E–F<br>PCI Express 3 (x2) (2.5 Gbps) → SerDes1 Lanes G–H |

[1] x8 PCI Express is only available at CCB (platform) speeds of 527 MHz or greater.

#### 4.4.3.9 SerDes2 I/O Port Selection

The device can be configured with different I/O ports active on SerDes2. Table 4-17 shows the configuration of I/O ports and bit rates (and required reference clocks) that are possible for SerDes2 interfaces.

**NOTE**

Any disabled SATA controller(s) will have their respective bit(s) set in DEVDISR (see Section 23.4.1.10, "Device Disable Register (DEVDISR)"), and therefore cannot respond to configuration accesses, as described in Section 23.5.1.5, "Shutting Down Unused Blocks."

**Table 4-17. SerDes2 I/O Port Selection**

| Functional Signal | Reset Configuration Name | Value (Binary) | Meaning |
|---|---|---|---|
| TSEC1_TXD2,<br>TSEC3_TXD2,<br>TSEC_1588_<br>PULSE_OUT1<br><br>Default (111) | cfg_srds2_prtcl[0:2] | 000 | Reserved |
| | | 001 | SATA1 → SerDes2 Lane A.<br>SATA2 → SerDes2 Lane B.<br>eTSEC1 and eTSEC3 Ethernet interface uses parallel interface according to POR config inputs cfg_tsec1_prtcl and cfg_tsec3_prtcl. |
| | | 010 | Reserved |
| | | 011 | SATA1 → SerDes2 Lane A.<br>SATA2 disabled.<br>eTSEC1 and eTSEC3 Ethernet interface uses parallel interface according to POR config inputs cfg_tsec1_prtcl and cfg_tsec3_prtcl.<br>SerDes2 Lane B disabled. |
| | | 100 | SATA1 and SATA2 disabled.<br>eTSEC1 SGMII (1.25 Gbps)→ SerDes2 Lane A.<br>eTSEC3 SGMII (1.25 Gbps) → SerDes2 Lane B.<br>POR config inputs cfg_tsec1_prtcl and cfg_tsec3_prtcl should be left in their default settings. |
| | | 101 | Reserved |
| | | 110 | SATA1 and SATA2 disabled.<br>eTSEC1 SGMII (1.25 Gbps) → SerDes2 Lane A (POR config input cfg_tsec1_prtcl should be left in its default setting)<br>eTSEC3 parallel mode Ethernet interface (according to cfg_tsec3_prtcl).<br>SerDes2 Lane B disabled |
| | | 111 | SATA1 and SATA2 disabled.<br>eTSEC1 and eTSEC3 Ethernet interface uses parallel interface according to POR config inputs cfg_tsec1_prtcl and cfg_tsec3_prtcl.<br>SerDes2 disabled |

#### 4.4.3.10 CPU Boot Configuration

The CPU boot configuration input, shown in Table 4-18, specifies the boot configuration mode. If LA27 is sampled low at reset, the e500 core is prevented from fetching boot code until configuration by an external master is complete. The external master frees the CPU to boot by setting EEBPCR[CPU_EN] in the ECM CCB port configuration register (EEBPCR). See Section 7.2.1.2, "ECM CCB Port Configuration Register (EEBPCR)," for more information.

Note that the value latched on this signal during POR is accessible through the memory-mapped PORBMSR (POR boot mode status register) described in Section 23.4.1.2, “POR Boot Mode Status Register (PORBMSR).”

Note also that the value latched on this signal during POR affects the PCI agent lock mode (See Section 16.3.2.19, “PCI Bus Function Register (PBFR).”) and the PCI Express Configuration Ready Register (See Section 17.3.10.18, “Configuration Ready Register—0x4B0.”).

**Table 4-18. CPU Boot Configuration**

| Functional Signal | Reset Configuration Name | Value (Binary) | Meaning |
|---|---|---|---|
| LA27<br>Default (1) | cfg_cpu_boot | 0 | CPU boot holdoff mode. The e500 core is prevented from booting until configured by an external master. |
| | | 1 | The e500 core is allowed to boot without waiting for configuration by an external master (default). |

### 4.4.3.11 Boot Sequencer Configuration

The boot sequencer configuration options, shown in Table 4-19, allow the boot sequencer to load configuration data from the serial ROM located on the I$^2$C1 port before the host tries to configure the MPC8536E. These options also specify normal or extended I$^2$C addressing modes. See Section 11.4.5, “Boot Sequencer Mode,” for more information on the boot sequencer.

Note that the values latched on these signals during POR are accessible through the memory-mapped PORBMSR (POR boot mode status register) described in Section 23.4.1.2, “POR Boot Mode Status Register (PORBMSR).”

**Table 4-19. Boot Sequencer Configuration**

| Functional Signal | Reset Configuration Name | Value (Binary) | Meaning |
|---|---|---|---|
| LGPL3/$\overline{\text{LFWP}}$, LGPL5<br>Default (11) | cfg_boot_seq[0:1] | 00 | Reserved |
| | | 01 | Normal I$^2$C addressing mode is used. Boot sequencer is enabled and loads configuration information from a ROM on the I$^2$C interface. A valid ROM must be present. |
| | | 10 | Extended I$^2$C addressing mode is used. Boot sequencer is enabled and loads configuration information from a ROM on the I$^2$C1 interface. A valid ROM must be present. |
| | | 11 | Boot sequencer is disabled. No I$^2$C ROM is accessed (default). |

**NOTE**

When the boot sequencer is enabled, the processor core will be held in reset and thus prevented from fetching boot code until the boot sequencer has completed its task, regardless of the state of the CPU boot configuration signal described in Section 4.4.3.10, “CPU Boot Configuration.”

### 4.4.3.12 DDR SDRAM Type

DDR3 requires a different voltage level from DDR2. Table 4-20 describes the configuration of the DDR SDRAM type.

Table 4-20. DDR DRAM Type

| Functional Signal | Reset Configuration Name | Value (Binary) | Meaning |
|---|---|---|---|
| LGPL0/LFCLE<br>Default (1) | cfg_dram_type | 0 | DDR3<br>1.5 V, CKE low at reset |
| | | 1 | DDR2<br>1.8 V, CKE low at reset (default) |

### 4.4.3.13 Serdes 2 Reference Clock Configuration

Three options are available for the frequency of the input SerDes2 reference clock: either a 100-MHz, 125-MHz, or 150-MHz LVDS differential clock.

This one clock is applied to an internal PLL whose output creates the clock used by all SGMII/SATA SerDes lanes. The result is always a 1.25-Gbaud transmission/receive rate on each lane when used for SGMII, and 1.5-Gbaud or 3.0-Gbaud when used for SATA. Note that the value latched on this signal during POR is accessible through the memory-mapped Section 23.4.1.6, “POR Device Status Register 2 (PORDEVSR2).”

Table 4-21. Serdes 2 Reference Clock Configuration

| Functional Signal | Reset Configuration Name | Value (Binary) | Meaning |
|---|---|---|---|
| TSEC3_TXD3, TSEC_1588_ PULSE_OUT2<br>Default (11) | cfg_srds2_ref_clk[0:1] | 00 | Reserved |
| | | 01 | When configured for SATA: SerDes2 expects a 150-MHz reference clock frequency.<br>This should not be used for when SerDes2 is configured for SGMII. |
| | | 10 | SerDes2 expects a 125-MHz reference clock frequency for either SATA or SGMII functionality. |
| | | 11 | SerDes 2 expects a 100-MHz reference clock frequency for either SATA or SGMII functionality (default). |

#### 4.4.3.14 eTSEC1 width

The eTSEC width input, shown in Table 4-22, selects standard versus reduced width for three-speed Ethernet controller interface 1. Note that the value latched on this signal during POR is accessible through the memory-mapped PORDEVSR (POR device status register) described in Section 23.4.1.4, “POR Device Status Register (PORDEVSR).”

**Table 4-22. eTSEC1 Width Configuration**

| Functional Signal | Reset Configuration Name | Value (Binary) | Meaning |
|---|---|---|---|
| TSEC1_TX_ER<br>Default (1) | cfg_tsec1_reduce | 0 | eTSEC1 interface operates in reduced pin mode, either RTBI, RGMII or RMII. |
| | | 1 | eTSEC1 interface operates in standard width TBI, GMII, MII, or in 8-bit FIFO mode. (default) |

This input does not affect the width of the eTSEC1 FIFO interface, which is always an 8-bit FIFO interface.

#### 4.4.3.15 eTSEC3 Width

The eTSEC3 width input, shown in Table 4-23, selects standard versus reduced width for three-speed Ethernet controller interface 3. Note that the value latched on this signal during POR is accessible through the memory-mapped PORDEVSR (POR device status register) described in Section 23.4.1.4, “POR Device Status Register (PORDEVSR).”

**Table 4-23. eTSEC3 Width Configuration**

| Functional Signal | Reset Configuration Name | Value (Binary) | Meaning |
|---|---|---|---|
| TSEC3_TX_ER<br>Default (1) | cfg_tsec3_reduce | 0 | eTSEC3 Ethernet interface operates in reduced mode, either RTBI, RGMII or RMII. |
| | | 1 | eTSEC3 Ethernet interface operates in standard TBI, GMII, MII, or 8-bit FIFO mode (default). |

The value of this configuration setting does not affect the width of the FIFO interface on eTSEC3, which is always 8 bits.

### 4.4.3.16 eTSEC1 Protocol

The eTSEC1 protocol inputs, shown in Table 4-24, select the protocol (FIFO, MII, GMII or TBI) used by the eTSEC1 controller. Note that the value latched on these signals during POR is accessible through the memory-mapped PORDEVSR (POR device status register) described in Section 23.4.1.4, "POR Device Status Register (PORDEVSR)."

Table 4-24. eTSEC1 Protocol Configuration

| Functional Signal | Reset Configuration Name | Value (Binary) | Meaning |
|---|---|---|---|
| TSEC1_TXD[0:1]<br>Default (11) | cfg_tsec1_prtcl[0:1] | 00 | The eTSEC1 controller operates using 8-bit FIFO protocol. |
| | | 01 | The eTSEC1 controller operates using the MII protocol (or RMII if configured in reduced mode as described in Section 4.4.3.14, "eTSEC1 width"). |
| | | 10 | The eTSEC1 controller operates using the GMII protocol (or RGMII if configured in reduced mode as described in Section 4.4.3.14, "eTSEC1 width"). |
| | | 11 | The eTSEC1 controller operates using the TBI protocol (or RTBI if configured in reduced mode as described in Section 4.4.3.14, "eTSEC1 width") (default). |

### 4.4.3.17 eTSEC3 Protocol

The eTSEC3 protocol inputs, shown in Table 4-25, select the protocol (FIFO, MII, GMII or TBI) used by the eTSEC3 controller. Note that the value latched on these signals during POR is accessible through the memory-mapped PORDEVSR (POR device status register) described in Section 23.4.1.4, "POR Device Status Register (PORDEVSR)."

Table 4-25. eTSEC3 Protocol Configuration

| Functional Signal | Reset Configuration Name | Value (Binary) | Meaning |
|---|---|---|---|
| TSEC3_TXD[0:1]<br>Default (11) | cfg_tsec3_prtcl[0:1] | 00 | The eTSEC3 controller operates using 8-bit FIFO protocol. |
| | | 01 | The eTSEC3 controller operates using the MII protocol (or RMII if configured in reduced mode as described in Section 4.4.3.15, "eTSEC3 Width") |
| | | 10 | The eTSEC3 controller operates using the GMII protocol (or RGMII if configured in reduced mode as described in Section 4.4.3.15, "eTSEC3 Width"). |
| | | 11 | The eTSEC3 controller operates using the TBI protocol (or RTBI if configured in reduced mode as described in Section 4.4.3.15, "eTSEC3 Width") (default). |

### 4.4.3.18 PCI Clock Selection

The PCI clock source inputs, shown in Table 4-26, specify the clock mode (synchronous or asynchronous) for the PCI interface. See Section Section 4.4.4.1, “System Clock/PCI Clock/DDR Clock” for more information. Note that the value latched on this signal during POR is accessible through the memory-mapped PORDEVSR (POR device status register) described in Section 23.4.1.4, “POR Device Status Register (PORDEVSR).”

**Table 4-26. PCI Clock Select**

| Functional Signal | Reset Configuration Name | Value (Binary) | Meaning |
|---|---|---|---|
| USB1_STP<br>Default (1) | cfg_pci_clk | 0 | Asynchronous mode. PCI_CLK is used as the clock for the PCI interface |
| | | 1 | Synchronous mode. SYSCLK is used as the clock for the PCI interface (default). |

### 4.4.3.19 PCI Speed Configuration

The PCI speed configuration input, shown in Table 4-27, configures internal logic for proper operation with the PCI clock frequencies in use. The default setting is appropriate for PCI operating above 33 MHz; for low speed operation (PCI at or below 33 MHZ) this POR configuration input should be low during $\overline{\text{HRESET}}$. If this configuration is not set properly, behavior of the PCI interface may be unreliable.

Note that the value latched on this signal during POR is accessible through the memory-mapped PORDEVSR, described in Section 23.4.1.4, “POR Device Status Register (PORDEVSR).”

**Table 4-27. PCI Speed Configuration**

| Functional Signal | Reset Configuration Name | Value (Binary) | Meaning |
|---|---|---|---|
| USB2_STP<br>Default (1) | cfg_pci_speed | 0 | PCI frequency at or below 33 MHz |
| | | 1 | PCI frequency above 33 MHz (default) |

### 4.4.3.20 PCI I/O Impedance

The PCI I/O impedance configuration inputs, shown in Table 4-28 select the impedance of the PCI I/O drivers for the respective interfaces. Note that the values latched on these signals during POR are accessible through PORIMPSCR, described in Section 23.4.1.3, “POR I/O Impedance Status and Control Register (PORIMPSCR).”

**Table 4-28. PCI I/O Impedance**

| Functional Signal | Reset Configuration Name | Value (Binary) | Meaning |
|---|---|---|---|
| $\overline{\text{PCI\_GNT1}}$<br>Default (1) | cfg_pci_impd | 0 | 25-Ω I/O drivers are used on the PCI interface. |
| | | 1 | 42-Ω I/O drivers are used on the PCI interface (default). |

#### 4.4.3.21 PCI Arbiter Configuration

The PCI arbiter configuration inputs, shown in Table 4-29, enable the on-chip PCI arbiter. Note that the value latched on these signals during POR are accessible through the PORDEVSR described in Section 23.4.1.4, "POR Device Status Register (PORDEVSR)."

Table 4-29. PCI Arbiter Configuration

| Functional Signal | Reset Configuration Name | Value (Binary) | Meaning |
|---|---|---|---|
| $\overline{\text{PCI\_GNT2}}$<br>Default (1) | cfg_pci_arb | 0 | The on-chip PCI arbiter is disabled. External arbitration is required. |
| | | 1 | The on-chip PCI arbiter is enabled (default). |

#### 4.4.3.22 Memory Debug Configuration

The memory debug configuration input, shown in Table 4-30, selects which debug outputs (DDR or LBC memory controller) are driven onto the MSRCID and MDVAL debug signals. Note that the value latched on this signal during POR is accessible through the memory-mapped PORDBGMSR (POR debug mode register) described in Section 23.4.1.5, "POR Debug Mode Status Register (PORDBGMSR)."

Table 4-30. Memory Debug Configuration

| Functional Signal | Reset Configuration Name | Value (Binary) | Meaning |
|---|---|---|---|
| MSRCID0<br>Default (1) | cfg_mem_debug | 0 | Debug information from the enhanced local bus controller (eLBC) is driven on the MSRCID and MDVAL signals |
| | | 1 | Debug information from the DDR SDRAM controller is driven on the MSRCID and MDVAL signals (default). |

#### 4.4.3.23 DDR Debug Configuration

The DDR debug configuration input, shown in Table 4-31, enables a DDR memory controller debug mode in which the DDR SDRAM source ID field and data valid strobe are driven onto the ECC pins. ECC checking and generation are disabled in this case. ECC signals driven from the SDRAMs must be electrically disconnected from the ECC I/O pins of the MPC8536E in this mode.

Table 4-31. DDR Debug Configuration

| Functional Signal | Reset Configuration Name | Value (Binary) | Meaning |
|---|---|---|---|
| MSRCID1<br>Default (1) | cfg_ddr_debug | 0 | Debug information is driven on the ECC pins instead of normal ECC I/O. ECC signals from memory devices must be disconnected. |
| | | 1 | Debug information is not driven on ECC pins. ECC pins function in their normal mode (default). |

Note that the value latched on this signal during POR is accessible through the memory-mapped PORDBGMSR (POR debug mode register) described in Section 23.4.1.5, "POR Debug Mode Status Register (PORDBGMSR)."

### 4.4.3.24 General-Purpose POR Configuration

The LBC address/data bus inputs, shown in Table 4-32, configure the value of the general-purpose POR configuration register defined in Section 23.4.1.7, "General-Purpose POR Configuration Register (GPPORCR)." This register is intended to facilitate POR configuration of user systems. A value placed on LAD[0:31] during POR is captured and stored (read only) in the GPPORCR. Software can then use this value to inform the operating system about initial system configuration. Typical interpretations include circuit board type, board ID number, or a list of available peripherals.

**Table 4-32. General-Purpose POR Configuration**

| Functional Signals | Reset Configuration Name | Value (Binary) | Meaning |
|---|---|---|---|
| LAD[0:31]<br>No default | cfg_gpinput[0:31] | — | General-purpose POR configuration vector to be placed in GPPORCR |

### 4.4.3.25 Engineering Use POR Configuration

These POR configuration inputs may be used in the future to control functionality. It is advised that boards are built with the ability to pull up or pull down these pins. Note that the value latched on these signals during POR are accessible through the PORDEVSR2, described in Section 23.4.1.6, "POR Device Status Register 2 (PORDEVSR2)."

**Table 4-33. Engineering Use POR Configuration**

| Functional Signal | Reset Configuration Name | Value (Binary) | Meaning |
|---|---|---|---|
| EC_MDC,<br>TSEC1_TXD3,<br>TSEC3_TXD7<br>Default (111) | cfg_eng_use[0:2] | 000–110 | Reserved |
| | | 111 | Default operation |

## 4.4.4 Clocking

The following paragraphs describe the clocking within the device.

### 4.4.4.1 System Clock/PCI Clock/DDR Clock

The MPC8536E takes a single input clock, SYSCLK, as its primary clock source for the e500 core and all of the devices and interfaces that operate synchronously with the core. As shown in Figure 4-6, the SYSCLK input (frequency) is multiplied up using a phase lock loop (PLL) to create the core complex bus (CCB) clock (also called the platform clock). The CCB clock is used by virtually all of the synchronous system logic, including the L2 cache, and other internal blocks such as the DMA and interrupt controller. The CCB clock also feeds the PLL in the e500 core and the PLL that create clocks for the local bus memory controller.

The PCI interface may use SYSCLK as the PCI clock and thus have PCI operation be synchronous with the platform. Alternately, a separate, independent clock may be used for the PCI interface, in which case PCI operation is asynchronous with respect to SYSCLK and the platform clock. If the separate (asynchronous) PCI_CLK clock signal is used rather than SYSCLK as the PCI clock, then this clock must be constantly driven, even when in Deep Sleep mode, in order to avoid loss of lock.

The DDR memory controller complex may use the platform clock and thus have operation of the DDR interface be synchronous with the platform. Alternately, an independent clock, DDRCLK, may be multiplied up using a separate PLL to create a unique DDR memory controller complex clock. In this case, the DDR complex operates asynchronously with respect to the platform clock.

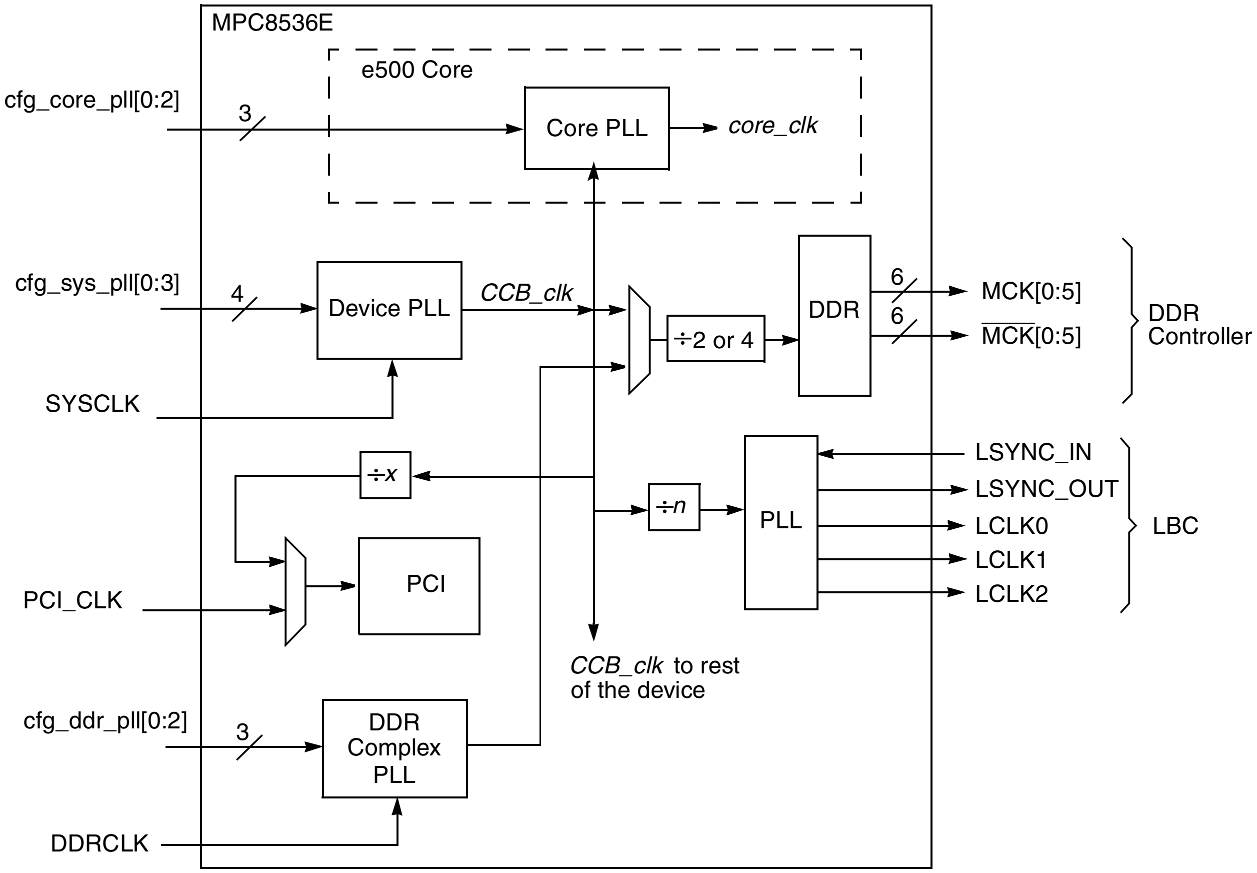


**Figure 4-6. Clock Subsystem Block Diagram**

### 4.4.4.2 PCI Express and SGMII Clocks

Clocks for these high speed interfaces on the MPC8536E are derived from a PLL in the SerDes block. This PLL is driven by a reference clock (SD*n*_REF_CLK/SD*n*_REF_CLK) whose input frequency is a function of the protocol and bit rate being used as shown in Table 4-34.

**Table 4-34. High Speed Interface Clocking**

| Interfaces | Bit Rate | Reference Clock Frequency |
|---|---|---|
| PCI Express | 2.5 Gbps | 100 MHz (Spread Spectrum supported) |
| SGMII | 1.25 Gbps | 100 MHz |

For any SerDes that is not disabled through cfg_io_ports[0:2]=001 or cfg_srds2_prtcl[0:2]=111 respectively, the applicable SD*n*_REF_CLK/$\overline{\mathrm{SD}n\_\mathrm{REF\_CLK}}$ must be constantly driven, even when in Deep Sleep mode, in order to avoid loss of lock.

#### 4.4.4.2.1 Minimum Frequency Requirements

Section 4.4.3.8, “SerDes1 I/O Port Selection,” describes various high-speed interface configuration options. Note that the CCB clock frequency must be considered for proper operation of such interfaces as described below.

For proper PCI Express operation, the CCB clock frequency must be greater than or equal to:

$$\frac{527\ \mathrm{MHz} \times (\mathrm{PCI\ Express\ link\ width})}{8}$$

See Section 17.1.3.2, “Link Width,” for PCI Express interface width details.

Note that the minimum CCB:SYSCLK ratio for PCI in synchronous mode is 6:1. See Section 4.4.3.1, “System PLL Ratio,” for details of selecting this ratio.

### 4.4.4.3 Ethernet Clocks

The Ethernet blocks operate asynchronously with respect to the rest of the device. These blocks use receive and transmit clocks supplied by their respective PHY chips, plus a 125-MHz clock input for gigabit protocols. Data transfers are synchronized to the CCB clock internally.

### 4.4.4.4 Real Time Clock

As shown in Figure 4-7, the real time clock (RTC) input can optionally be used to clock the e500 core timer facilities. RTC can also be used (optionally) by the programmable interrupt controller (PIC) global timer facilities. The RTC is separate from the e500 core clock and is intended to support relatively low frequency timing applications. The RTC frequency range is specified in the *MPC8536E Integrated Processor Hardware Specifications*, but the maximum value should not exceed one-quarter of the CCB Frequency.

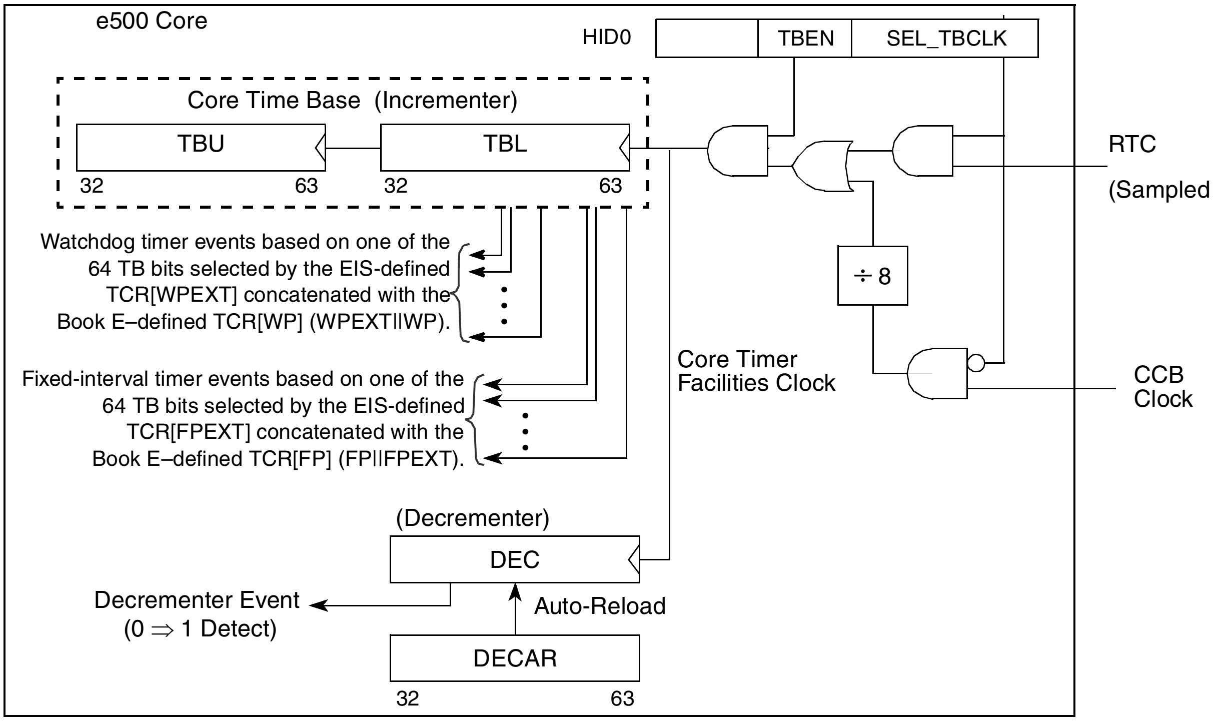
Before being distributed to the core time base, RTC is sampled and synchronized with the CCB clock.

The clock source for the core time base is specified by two fields in HID0: time base enable (TBEN), and select time base clock (SEL_TBCLK). If the time base is enabled, (HID0[TBEN] is set), the clock source is determined as follows:

- HID0[SEL_TBCLK] = 0, the time base is updated every 8 CCB clocks
- HID0[SEL_TBCLK] = 1, the time base is updated on the rising edge of RTC

The default source of the time base is the CCB clock divided by eight. For more details, see the *PowerPC e500 Core Complex Reference Manual*.

Section 9.3.2.6, “Timer Control Registers (TCRA–TCRB),” provides additional information on the use of the RTC signal to clock the global timers in the PIC unit.


e500 Core
HID0
TBEN
SEL_TBCLK
Core Time Base (Incrementer)
TBU
TBL
32
63
32
63
RTC
(Sampled
÷ 8
Watchdog timer events based on one of the 64 TB bits selected by the EIS-defined TCR[WPEXT] concatenated with the Book E–defined TCR[WP] (WPEXT||WP).
Fixed-interval timer events based on one of the 64 TB bits selected by the EIS-defined TCR[FPEXT] concatenated with the Book E–defined TCR[FP] (FP||FPEXT).
Core Timer Facilities Clock
CCB Clock
(Decrementer)
DEC
Decrementer Event (0 ⇒ 1 Detect)
Auto-Reload
DECAR
32
63

**Note:** The logic circuits shown depict functional relationships only; they do not represent physical implementation details.

**Figure 4-7. RTC and Core Timer Facilities Clocking Options**

## 4.5 Initialization/Applications Information

### 4.5.1 System Boot

Selecting on-chip ROM in boot ROM location, see Table 4-14, causes the e500 CPU to fetch data from the on-chip ROM. The on-chip ROM is selected by configuring the POR config pins *cfg_rom_loc[0:3]*. Two different configurations are provided for boot from the on-chip ROM - boot from eSPI and from eSDHC.

#### 4.5.1.1 eSDHC Boot

##### 4.5.1.1.1 Overview

The MPC8536E is capable of loading initialization code from a memory device that is connected to the eSDHC controller interface. This device can be either a SD or MMC card or other variants compatible with these devices. The term SD/MMC will be used when referring to the memory device.

Boot from eSDHC is supported by the MPC8536E using an on-chip ROM which contains the basic eSDHC device driver and the code to perform block copy from SD/MMC to any target memory. Selecting on-chip ROM in boot ROM location (see Table 4-14) causes the e500 CPU to fetch data from the on-chip ROM. The on-chip ROM is selected by configuring the POR config pins *cfg_rom_loc[0:3]*. Prior to boot, the user must ensure that the SD/MMC card to boot from is inserted.

After the device has completed the reset sequence, if the ROM location selects the on-chip ROM eSDHC Boot configuration, the e500 core starts to execute code from the internal on-chip ROM. The e500 core configures the eSDHC controller, enabling it to communicate with the external SD/MMC card. The SD/MMC card should contain a specific data structure with control words, device configuration information and initialization code. The on-chip ROM boot code uses the information from the SD/MMC card content to configure the device, and to copy the initialization code to a target memory device (for example, the DDR) through the eSDHC interface. After all the code has been copied, the e500 core starts to execute the code from the target memory device.

There are several different ways a user may utilise the eSDHC boot feature. The simplest is for the on-chip ROM to copy an entire operating system boot image into system memory, and then jump to it to begin execution. However, this may be many megabytes and in some situations may be sub-optimal, since only 1-bit mode is used during booting.

A more advanced option is for the on-chip ROM to only copy a small user-customised subroutine, which configures the eSDHC in an optimal way. The user-customised subroutine then copies the rest of the boot code potentially much faster than the on-chip ROM software can achieve. For example, the user-customised subroutine may utilise 4-bit or 8-bit eSDHC interfaces, or support new SD or MMC format revisions, or increase the external clock frequency based on knowledge of the exact frequency that the MPC8536E is operating at.

#### 4.5.1.1.2 Features

- Provides mechanism to load initialization code from the following external devices:
  - SD memory cards, including the memory portion of SD Combo cards (up to and including version 2.0)
  - MMC, RS-MMC and MMC*plus* (up to and including version 4.0)
  - SDHC cards (SD High Capacity, from 4GByte to 32GByte)
- Boot from the following devices is not supported
  - SDIO and miniSDIO cards which are not SD Combo cards and consequently have no memory
  - Locked (password-protected) SD/MMC cards
  - Secured Mode of SD cards (SD Card Specification Part 3: Security Specification)
- Simple data structure in SD/MMC card
- BOOT signature will be checked to validate that the SD/MMC card contains valid code
- Supports variable code length in SD/MMC card
- Flexible target memory device
- Supports target memory configuration controlled by the user
- Only 1-bit operation is supported for boot (even if the SD/MMC card supports 4 or 8-bit parallel access).
- Initial setting will use a serial clock below 400 kHz; the SD/MMC internal registers are read by initialization code and parsed to determine the optimal clock frequency supported by the SD/MMC card inserted.
- High speed cards are supported (up to 50MHz SD and 52MHz MMC).

- Control word will allow for user modification
- There must be precisely one device connected on the eSDHC bus. In particular, multiple MMC devices sharing the one bus are not supported.
- Compatible with FAT12/FAT16/FAT32 SD/MMC filesystems (provided <=40 Configuration Words are used prior to copying user's code to system memory).
- Redundancy to support SD/MMC bad blocks, by searching for the BOOT signature in up to 24 blocks, and trying the next block if the BOOT signature is not found, or if a read CRC error is found.

#### 4.5.1.1.3 SD/MMC Card Data Structure

The SD/MMC card should contain the initialization code length in bytes, source address in the SD/MMC card, destination address in the target memory device, execution starting address, and multiple configuration words with pairs of target address and its respective data.

Figure 4-11 shows the required SD/MMC card data structure.

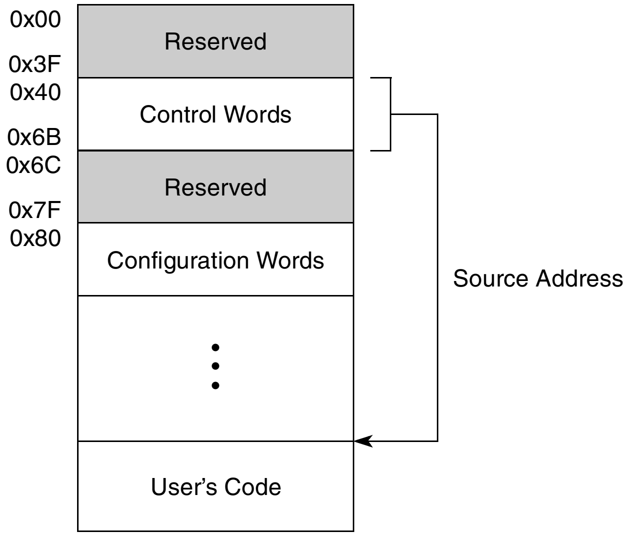


**Figure 4-8. SD/MMC Card Data Structure**

**Table 4-35. SD/MMC Card Data Structure**

| Address | Data Bits [0:31] |
|---|---|
| 0x00–0x3F | Reserved. |
| 0x40–0x43 | BOOT signature. This location should contain the value 0x424f_4f54, which is the ascii code for BOOT. The boot loader code will search for this signature. If the value in this location doesn't match the BOOT signature, it means that the SD/MMC card doesn't contain a valid user code. In such case the boot loader code will disable the eSDHC and will issue a hardware reset request of the SoC by setting RSTCR[HRESET_REQ]. |
| 0x44–0x47 | Reserved |

### Table 4-35. SD/MMC Card Data Structure (continued)

| Address | Data Bits [0:31] |
|---|---|
| 0x48–0x4B | User’s code length. Number of bytes in the user’s code to be copied. This must be a multiple of the SD/MMC card’s block size (and the user’s code zero-padded if necessary to achieve that length).<br>User’s code length <= 2GBytes. |
| 0x4C–0x4F | Reserved |
| 0x50–0x53 | Source Address. Contains the starting address of the user’s code as an offset from the SD/MMC card starting address.<br><br>In Standard Capacity SD/MMC Cards, the 32-bit Source Address specifies the memory address in byte address format. This must be a multiple of the SD/MMC card’s block size.<br>In High Capacity SD Cards (>2GByte), the 32-bit Source Address specifies the memory address in block address format. Block length is fixed to 512 bytes as per the SD High Capacity specification. |
| 0x54–0x57 | Reserved |
| 0x58–0x5B | Target Address. Contains the target address in the system’s local memory address space in which the user’s code will be copied to. This is a 32-bit effective address. The core is configured in such a way that the 36-bit real address is equal to this (with 4 most significant bits zero). |
| 0x5C–0x5F | Reserved |
| 0x60–0x63 | Execution Starting Address. Contains the jump address in the system’s local memory address space into the user’s code first instruction to be executed. This is a 32-bit effective address. The core is configured in such a way that the 36-bit real address is equal to this (with 4 most significant bits zero). |
| 0x64–0x67 | Reserved |
| 0x68–0x6B | N. Number of Config Address/Data pairs.<br>Must be 1<=N<=1024 (but is recommended to be as small as possible). |
| 0x6C–0x7F | Reserved. |
| 0x80–0x83 | Config Address 1 |
| 0x84–0x87 | Config Data 1 |
| 0x88–0x8B | Config Address 2 |
| 0x8C–0x8F | Config Data 2 |
| ... | |
| 0x80 + 8*(N–1) | Config Address N[1] |
| 0x80 + 8*(N–1)+4 | Config Data N (final Config Data N optional) |
| ~<br>~<br>~ | |
| | User’s code. Note that user's code must start on a 512-byte boundary. |

[1] N <=40 if compatibility with FAT12/FAT16/FAT32 filesystems is required. Refer to Section , “Notes on compatibility with FAT12/FAT16/FAT32 Filesystems” for details.

### Configuration Words Section

The configuration words section is comprised of Config Address and Config Data pairs of adjacent 32-bit fields. These are typically used to configure the local access windows and the target memory controller's registers. They are therefore system-dependent, as they need to be aware of the type and configuration of memory in a particular system.

The Config Address field has two modes that are selected by the least significant bit in the field (CNT). If the CNT bit is clear, then the 30 most significant bits are used to form the address pointer and the Config Data contains the data to be written to this address. If the CNT bit is set then the 30 most significant bits are used for control instruction. This flexible structure allows the user to configure any 4-byte aligned memory mapped register, perform control instructions, and specify the end of the configuration stage.

Note that it is illegal to change the content of the CCSRBAR by using this mechanism. Any attempt to do so will cause the boot process to hang.

The upper 4 most-significant address bits of the 36-bit address are always zero. Consequently the configuration words can only access memory in the lowest 4 GByte segment of memory. However, since by default CCSRBAR maps all memory mapped registers within the lowest 4 GByte segment of memory, and the user is prohibited from changing CCSRBAR with a configuration access, this is not an issue.

The Config Address structure is shown in Figure 4-12.

| | 0 | 1 | 2 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|
| **CNT = 0** | Address | | | | — | CNT |
| **CNT = 1** | EC | DLY | — | | | CNT |

**Figure 4-9. Config Address Fields**

defines the Config Address bits when CNT = 0 (address mode).

**Table 4-36. Config Address Field Description, CNT = 0**

| Bits | Name | Description |
|---|---|---|
| 0–29 | Address | Address bits 0–29. The data in the Config Data field is copied by the e500 core to this address. The two least significant bits of the address (30:31) are always considered to be zero, as are the upper 4 bits of the 36-bit address. |
| 30 | — | Reserved. Must be zero. |
| 31 | CNT | Control. Select between Address mode and Control mode.<br>0 Address mode<br>1 Control mode |

**Table 4-37. Config Address Field Description, CNT = 1**

| Bits | Name | Description |
|---|---|---|
| 0 | EC | End Configuration. Indicates the end of the configuration stage. Valid only if bit CNT is set.<br>0 Not the last Config Address field.<br>1 The Last Config Address field. The e500 core will stop the configuration stage and start to copy the user's code.<br>This must be set for Config Address Word N, and not be set for Config Address words prior to Config Address Word N. |
| 1 | DLY | Delay. Instruct the e500 core to perform delay according to the number that is specified in the adjacent Config Data field. The adjacent Config Data field provides the delay measured in terms of the number of 8 CCB clocks. Valid only if bit CNT is set.<br>0 No delay.<br>1 Delay. |
| 2–29 | — | Reserved. Must be zero. |
| 30 | — | Reserved. Must be zero. |
| 31 | CNT | Control. Select between Address mode and Control mode.<br>(0 Address mode)<br>1 Control mode<br>**Note:** When CNT=1, bits 0–29 select the control instruction. Only one bit in the range of bits 0–29 can be set at any specific control instruction. A control instruction with bits 0–29 all cleared is also illegal. |

## Notes on compatibility with FAT12/FAT16/FAT32 Filesystems

Depending upon application, compatibility may be desired between the SD/MMC Card data structure defined here and the FAT12, FAT16 or FAT32 filesystems (documented in SD Card Specifications Part 2—File System Specification v2.0, among other places). This compatibility is possible, but imposes a limit on the number of Configuration Words that can be parsed by the processor prior to fetching the user's code. Compatibility is achieved by ensuring that the entire data structure of Control Words and Configuration Words is contained within the first 446-byte (0x1BE) Master Boot Record code area of the filesystem.

Given that Configuration Words start at address 0x80, and all Configuration Words (except the last one with EC=1 to end the configuration) occupy 8 bytes per Configuration Address/Data pair, this imposes the limit of a maximum of 40 Configuration Address words. More Configuration Words can be used in applications for which compatibility with the FAT Master Boot Record is not required. If exactly 40 Configuration Address words are used and FAT12/FAT16/FAT32 compatibility is required, then the final Configuration Data word must be omitted to ensure that the data structure fits in less than 446 bytes.

Note that FAT12, FAT16 and FAT32 standards impose additional requirements on the data structures that must be present on the SD/MMC card, such as Partition Tables and a fixed Signature Word at the end of the Master Boot Record. These features are not interpreted or required by the eSDHC boot process, and are outside the scope of this document.

Furthermore, FAT12 and FAT16 define a boot sector with defined fields in the first 0x36 addressable bytes (which does not conflict with the SD/MMC Card Data Structure for boot from SD/MMC defined in this document). Therefore FAT12 and FAT16 filesystems are completely compatible with the defined data structure, even if they also contain a FAT boot sector. However, FAT32 defines a boot sector with defined fields in the first 0x52 addressable bytes. Therefore, FAT32 filesystem compatibility is only possible if used in a system in which this boot sector information is not required.

Also note that the user code is copied from one sequential area of SD/MMC card memory space specified by the Source Address. The boot ROM software does not look for or parse any File Allocation Table, and furthermore, the boot ROM software assumes that the User Code is in one contiguous range of memory addresses.

#### 4.5.1.1.4 eSDHC Controller Initial Configuration

The eSDHC controller configuration is used by the boot ROM software. After the boot from eSDHC has finished, the user can change this configuration for other uses of the eSDHC interface. The boot ROM software also changes some of this configuration automatically depending upon the features supported by the SD/MMC card that is connected.

The eSDHC controller is initially configured to operate in the following configuration:

- Address Invariant Mode (eSDHC.PROTCTL[EMODE]=10)
- SDHC_DAT[3] does not monitor card insertion. The GPIO4/SDHC_CD pin is used for card detect (eSDHC.PROTCTL[D3CD]=0 and Global_Utilities.PMUXCR[SDHC_CD]=1).
- 1-bit mode (eSDHC.PROTCTL[DTW]=00)
- SDCLK at 400 kHz or below, but higher than 100 kHz (for MPC8536E platform frequency up to 66 MHz, and therefore eSDHC base clock frequency up to 333 MHz). This is done with eSDHC.SYSCTL[SCLKFS]=0x20 and eSDHC.SYSCTL[DVS]=0xC, for a divisor of 832.
- There must be precisely one device connected on the eSDHC bus (and this device must be inserted prior to boot). Multiple MMC devices sharing the one bus are not supported.
- The bus operates in push-pull mode (MPC8536E pads drive both logic “0” and logic “1” as appropriate). If a MMC card is to be connected, then weak external pull-ups are required on the SDHC_CMD and SDHC_DAT[] pins in order to interface with the MMC open-drain mode during initialisation.
- The eSDHC DMA engine is not used for Control or Configuration Word accesses; instead, all eSDHC data transfers are initiated by the processor core polling eSDHC.PRSSTAT[BRR] and accessing data through the DATPORT register (XFERTYP[DMAEN]=0). The eSDHC DMA engine is used for User Code accesses.

#### 4.5.1.1.5 eSDHC Controller Boot Sequence

The code in the eSDHC Boot ROM configuration performs the following sequence of events:

1. The eSDHC controller is configured as per Section 4.5.1.1.4, “eSDHC Controller Initial Configuration”.
2. Card-detect.
3. The SD/MMC card is reset.
4. SD/MMC card voltage validation is performed.
5. SD/MMC card identification.
6. With CMD9, the CSD (Card-Specific Data) register of the SD/MMC card is read.

7. Based on the values returned from the SD/MMC card's CSD register, the eSDHC's registers are updated to reflect the maximum clock frequency jointly supported by the eSDHC controller, and the SD/MMC card connected to it.
8. The eSDHC begins reading the SD/MMC data structure from the card.
9. The eSDHC begins fetching the user code from the card.
10. If either the BOOT signature is not found at memory offset 0x40, or if when reading the Control and Configuration Words or the User's Code a read CRC error is detected, then it may be due to a bad block on the SD/MMC memory card. To counteract this and provide error resilience, if this occurs the eSDHC returns to step 8, fetching data from an address 0x200 greater than the previously fetched address. For example, if there have been *i* failed attempts, then on the following try the BOOT signature is checked at offset 0x40+i*0x200. If this sequence fails 24 times, then the system boot is deemed to have failed.
11. The processor core waits until the user code DMA transfer is complete.
12. The processor core jumps to the Execution Starting Address to begin execution of the user's code.

#### 4.5.1.1.6 eSDHC Boot Error Handling

If at any stage the boot loader code detects an error and cannot continue, it will disable the eSDHC and will issue a hardware reset request of the SoC by setting RSTCR[HRESET_REQ]. This may occur in any of the following scenarios:

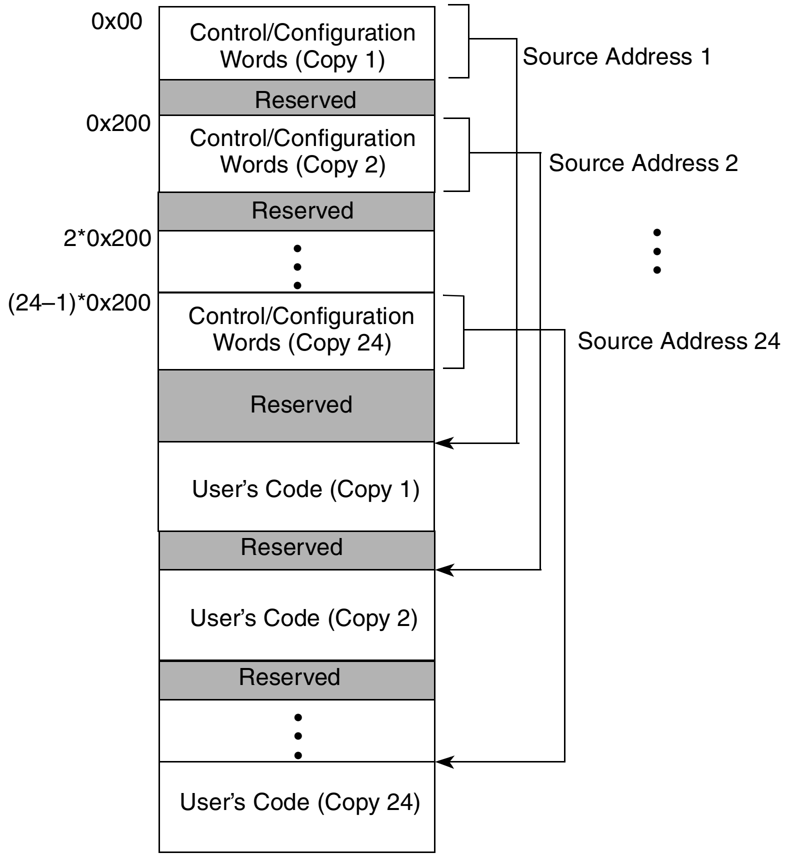
- BOOT signature not found at offset 0x40 or CRC error on any of the data read by the eSDHC 24 times.
- Timeout while waiting for the SD/MMC card to respond at any stage.
- No card inserted.
- Incorrect type of card inserted that is not supported for boot (such as CE-ATA).
- There is no common protocol, voltage or frequency mutually supported by the SD/MMC card and the eSDHC.
- The eSDHC reads as far as the Source Address (specified by the Control Word of the SD/MMC data structure) without seeing a EC=1 Configuration Word.

The boot loader code supports redundancy, which allows boot to succeed even in the presence of SD/MMC bad blocks. It does this by searching for the BOOT signature in up to 24 locations, and trying the next block if the BOOT signature is not found, or if a read CRC error is found. Each location tried is at a fixed offset of 512 bytes (0x200) from the previous (unsuccessful) offset, irrespective of the actual block size of the SD/MMC card.

For reference, the following diagram gives an example SD/MMC memory card data structure which can be used for maximum SD/MMC card data redundancy.

Note that if 0x40+8*(N–1)+4>=0x200 (where N is the number of Configuration Words), then care needs to be taken to ensure that the configuration words at 0x40+i*0x200 (for all 2<=i<=24) must not contain the BOOT signature. This ensures that the boot loader code does not mistakenly detect a BOOT signature. This also reduces the number of copies of boot code that can be used on one device.

Each copy of the Control/Configuration Words would generally be identical except for the pointer to the Source Address (offset 0x50) of the SD/MMC card, which may be different for each copy. If the User's Code section is sufficiently large that 24 copies of it do not fit in the capacity of the SD/MMC card (or if the SD/MMC card capacity must also be utilised for functional features other than system boot), then it may still be desirable to still support up to 24 copies of the Control/Configuration Words, but only have them reference a limited number of User's Code sections. This is also possible.



Note: User's code must begin on 512-byte boundary and its length must be a multiple of 512 bytes.

**Figure 4-10. SD/MMC Card Data Structure for Maximum Redundancy**

### 4.5.1.2 eSPI Boot ROM

#### 4.5.1.2.1 Overview

The MPC8536E is capable of loading initialization code from a memory device that is connected to the eSPI controller interface. This device can be either an EEPROM or a serial flash with an eSPI-compatible interface. The term EEPROM will be used when referring to the memory device.

The MPC8536E eSPI controller supports RapidS™ full clock cycle operation and Winbond™ dual-output read serial interface, but these modes are not enabled for boot by the on-chip ROM.

Boot from eSPI is supported by the MPC8536E using an on-chip ROM which contains the basic eSPI device driver and the code to perform block copy from eSPI EPROM to any target memory. Selecting on-chip ROM in boot ROM location (see Table 4-14) causes the e500 CPU to fetch data from the on-chip ROM. The on-chip ROM is selected by configuring the POR config pins *cfg_rom_loc[0:3]*.

After the device has completed the reset sequence, if the ROM location selects the on-chip ROM eSPI Boot configuration, the e500 core starts to execute code from the internal on-chip ROM. The e500 core configures the eSPI controller, enabling it to communicate with the external EEPROM. The EEPROM should contain a specific data structure with control words, device configuration information and initialization code. The on-chip ROM boot code uses the information from the EEPROM content to configure the device, and to copy the initialization code to a target memory device (for example, the DDR) through the eSPI interface. After all the code has been copied, the e500 core starts to execute the code from the target memory device.

There are several different ways a user may utilise the eSPI boot feature. The simplest is for the on-chip ROM to copy an entire operating system boot image into system memory, and then jump to it to begin execution. However, this may be many megabytes and in some situations may sub-optimal.

A more advanced option is for the on-chip ROM to only copy a small user-customised subroutine, which configures the eSPI in an optimal way. The user-customised subroutine then copies the rest of the boot code potentially much faster than the on-chip ROM software can achieve. For example, the user-customised subroutine may utilise Atmel RapidS or Winbond dual output eSPI modes.

#### 4.5.1.2.2 Features

- Provides mechanism to load initialization code from external eSPI EEPROM
- Simple data structure in eSPI EEPROM
- BOOT signature will be checked to validate that the EEPROM contains valid code
- Supports variable code length in EEPROM
- Flexible target memory device
- Supports target memory configuration controlled by the user
- Supports standard eSPI interface EEPROMs with read instruction code 0x03 followed by a 2-byte address (16-bit addressable EEPROMs) or 3-byte address (24-bit addressable EEPROMs).
- Initial setting will generate a serial clock below 5 MHz; the control word will allow for user modification of clock frequency.

#### 4.5.1.2.3 EEPROM Data Structure

The EEPROM should contain the initialization code length in bytes, source address in the eSPI EEPROM, destination address in the target memory device, execution starting address, and multiple configuration words with pairs of target address and its respective data.

Figure 4-11 shows the required eSPI EEPROM data structure.

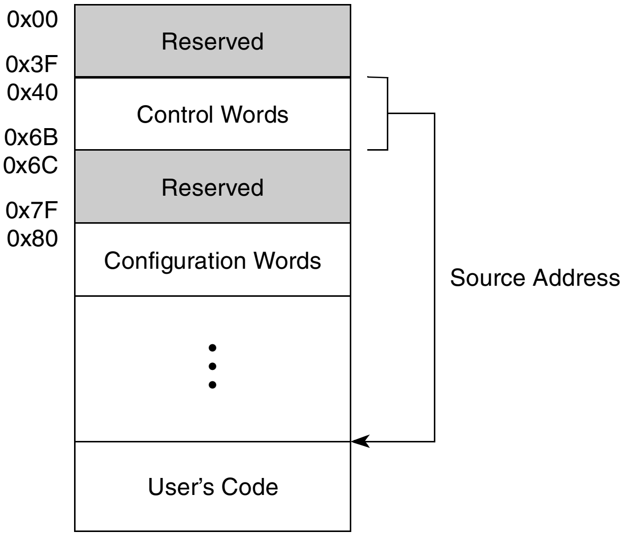


**Figure 4-11. eSPI EEPROM Data Structure**

**Table 4-38. eSPI EEPROM Data Structure**

| Address | Data Bits [0:31] |
|---|---|
| 0x00–0x3F | Reserved. |
| 0x40–0x43 | BOOT signature. This location should contain the value 0x424f_4f54, which is the ascii code for BOOT. The eSPI loader code will search for this signature, initially in 24-bit addressable mode. If the value in this location doesn't match the BOOT signature, then the EEPROM is accessed again, but in 16-bit mode. If the value in this location still doesn’t match the BOOT signature, it means that the eSPI device doesn't contain a valid user code. In such case the eSPI loader code will disable the eSPI and will issue a hardware reset request of the SoC by setting RSTCR[HRESET_REQ]. |
| 0x44–0x47 | Reserved |
| 0x48–0x4B | User’s code length. Number of bytes in the user’s code to be copied.<br>Must be a multiple of 4.<br>4<=User’s code length <= 2GBytes. |
| 0x4C–0x4F | Reserved |
| 0x50–0x53 | Source Address. Contains the starting address of the user’s code as an offset from the EEPROM starting address. In 24-bit addressing mode, the 8 most significant bits of this should be written to as zero, because the EEPROM is accessed with a 3-byte (24-bit) address. In 16-bit addressing mode, the 16 most significant bits of this should be written to as zero. |
| 0x54–0x57 | Reserved |
| 0x58–0x5B | Target Address. Contains the target address in the system’s local memory address space in which the user’s code will be copied to. This is a 32-bit effective address. The core is configured in such a way that the 36-bit real address is equal to this (with 4 most significant bits zero). |
| 0x5C–0x5F | Reserved |

Table 4-38. eSPI EEPROM Data Structure (continued)

| Address | Data Bits [0:31] |
|---|---|
| 0x60–0x63 | Execution Starting Address. Contains the jump address in the system's local memory address space into the user's code first instruction to be executed. This is a 32-bit effective address. The core is configured in such a way that the 36-bit real address is equal to this (with 4 most significant bits zero). |
| 0x64–0x67 | Reserved |
| 0x68–0x6B | N. Number of Config Address/Data pairs.<br>Must be <=1024 (but is recommended to be as small as possible). |
| 0x6C–0x7F | Reserved. |
| 0x80–0x83 | Config Address 1 |
| 0x84–0x87 | Config Data 1 |
| 0x88–0x8B | Config Address 2 |
| 0x8C–0x8F | Config Data 2 |
| ... | |
| 0x80 + 8*(N–1) | Config Address N |
| 0x80 + 8*(N–1)+4 | Config Data N (Final Config Data N optional) |
| ~<br>~<br>~ | |
| | User's Code |

## Configuration Words Section

The configuration words section is comprised of Config Address and Config Data pairs of adjacent 32-bit fields. These are typically used to configure the local access windows and the target memory controller's registers. They are therefore system-dependent, as they need to be aware of the type and configuration of memory in a particular system.

The Config Address field has two modes that are selected by the least significant bit in the field (CNT). If the CNT bit is clear, then the 30 most significant bits are used to form the address pointer and the Config Data contains the data to be written to this address. If the CNT bit is set then the 30 most significant bits are used for control instruction. This flexible structure allows the user to configure any 4-byte aligned memory mapped register, perform control instructions, and specify the end of the configuration stage.

Note that it is illegal to change the content of the CCSRBAR by using this mechanism. Any attempt to do so will cause the boot process to hang.

The upper 4 most-significant address bits of the 36-bit address are always zero. Consequently the configuration words can only access memory in the lowest 4 GByte segment of memory. However, since by default CCSRBAR maps all memory mapped registers within the lowest 4 GByte segment of memory, and the user is prohibited from changing CCSRBAR with a configuration access, this is not an issue.

The Config Address structure is shown in Figure 4-12.

| | 0 | 1 | 2 | 3 29 | 30 | 31 |
|---|---|---|---|---|---|---|
| **CNT = 0** | Address | | | | — | CNT |
| **CNT = 1** | EC | DLY | CF | — | — | CNT |

**Figure 4-12. Config Address Fields**

**Table 4-39. Config Address Field Description, CNT = 0**

| Bits | Name | Description |
|---|---|---|
| 0–29 | Address | Address bits 0–29. The data in the Config Data field is copied by the e500 core to this address. The two least significant bits of the address (30:31) are always considered to be zero, as are the upper 4 bits of the 36-bit address. |
| 30 | — | Reserved. Must be zero. |
| 31 | CNT | Control. Select between Address mode and Control mode.<br>0 Address mode<br>1 Control mode |

**Table 4-40. Config Address Field Description, CNT = 1**

| Bits | Name | Description |
|---|---|---|
| 0 | EC | End Configuration. Indicates the end of the configuration stage. Valid only if bit CNT is set.<br>0 Not the last Config Address field.<br>1 The Last Config Address field. The e500 core will stop the configuration stage and start to copy the user's code.<br>This must be set for Config Address Word N, and not be set for Config Address words prior to Config Address Word N. |
| 1 | DLY | Delay. Instruct the e500 core to perform delay according to the number that is specified in the adjacent Config Data field. The adjacent Config Data field provides the delay measured in terms of the number of 8 CCB clocks. Valid only if bit CNT is set.<br>0 No delay.<br>1 Delay. |
| 2 | CF | Change frequency. Instruct the e500 core to perform sequence of operations to setup the eSPI CS1 mode register with the frequency related (PM and DIV16) bits as defined by the user. The adjacent Config Data field will be written to the eSPI mode register. Software will use DIV16 and PM bits and mask all other bits such that they will not change. Software will perform the necessary steps which are required by the eSPI controller before and after changing the eSPI mode register.<br>This only takes effect after all of the Configuration and Control words have been read. |
| 3–29 | — | Reserved. Must be zero. |
| 30 | — | Reserved. Must be zero. |
| 31 | CNT | Control. Select between Address mode and Control mode.<br>(0 Address mode)<br>1 Control mode<br>**Note:** When CNT=1, bits 0–29 select the control instruction. Only one bit in the range of bits 0–29 can be set at any specific control instruction. A control instruction with bits 0–29 all cleared is also illegal. |

#### 4.5.1.2.4 eSPI Controller Configuration

The eSPI controller configuration is used by the eSPI boot ROM software. After the boot from eSPI has finished, the user can change this configuration for other uses of the eSPI interface.

The eSPI controller is configured to operate in master mode. The eSPI chip select 0 ($\overline{\text{SPI\_CS}}$[0]) must be connected to the EEPROM $\overline{\text{CS}}$ and selectively enables the EEPROM.

Figure 4-13 shows the external signal connection.

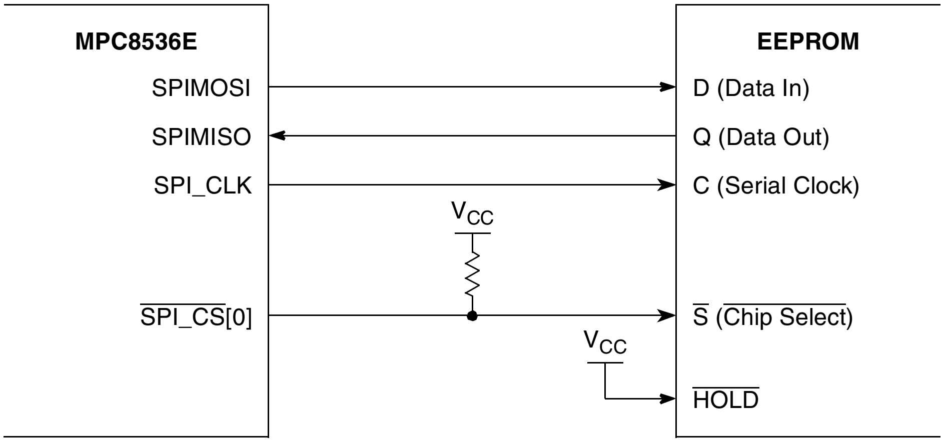


**Figure 4-13. External Signal Connection**

The eSPI controller is configured by the on-chip ROM code. The controller is configured as follows:

- Data is shifted out data on SPIMOSI during the falling edge of SPI_CLK. It samples data in from SPIMISO during the rising edge of SPI_CLK.
- The clock is low when the line is idle.
- It uses 8-bit length characters.
- The platform clock is divided by 256. For example, when the platform clock is configured to 533 MHz, the SPI_CLK will run at 2.08 MHz. (Note that frequency setting can be changed by using the CF control word, as explained in the "Configuration Words Section."
- The MSB is sent and received first.

Figure 4-14 shows the default eSPI CS0 mode register (SPMODE0) configuration.

|  | 0 | 1 | 2 | 3 | 4 7 | 8 | 9 10 | 11 | 12 15 | 16 19 | 20 23 | 24 28 | 29 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | CI0 | CP0 | REV0 | DIV160 | PM0 | ODD0 | — | POL0 | LEN0 | CS0BEF | CS0AFT | CS0CG | — |
| Value | 0 | 0 | 1 | 1 | 0 0 0 1 | 0 | 0 0 | 0 | 0 1 1 1 | 0 0 1 0 | 0 0 1 0 | 0 0 0 1 0 | 0 0 0 |

**Figure 4-14. eSPI CS0 Mode Register (SPMODE0) Configuration**

The ROM code will initially use the eSPI controller to generate standard read instruction code 0x03 followed by a 3-byte address for every non-sequential read operation (reading from a location which is not sequential to the last byte read). For sequential read operation, toggling the eSPI clock will cause the eSPI EEPROM to present the content of the next address location. The serial EEPROM must have an eSPI compatible interface with read instruction code 0x03 followed by a 2- or 3-byte address.

16-bit addressable EEPROM memories are supported and detected automatically by the boot code. This is accomplished by the boot code trying 16-bit mode if it fails to find the “BOOT” signature when in 24-bit mode.

Figure 4-15 shows the read instruction timing diagram for normal (not Atmel RapidS or Winbond Dual Output) modes of operation with a 24-bit addressable eSPI memory. With 16-bit addressable eSPI memories, only a 16-bit address is transmitted, and valid data is received from the EEPROM on the 24th SPI_CLK cycle rather than the 32nd SPI_CLK cycle for 24-bit addressable memories.

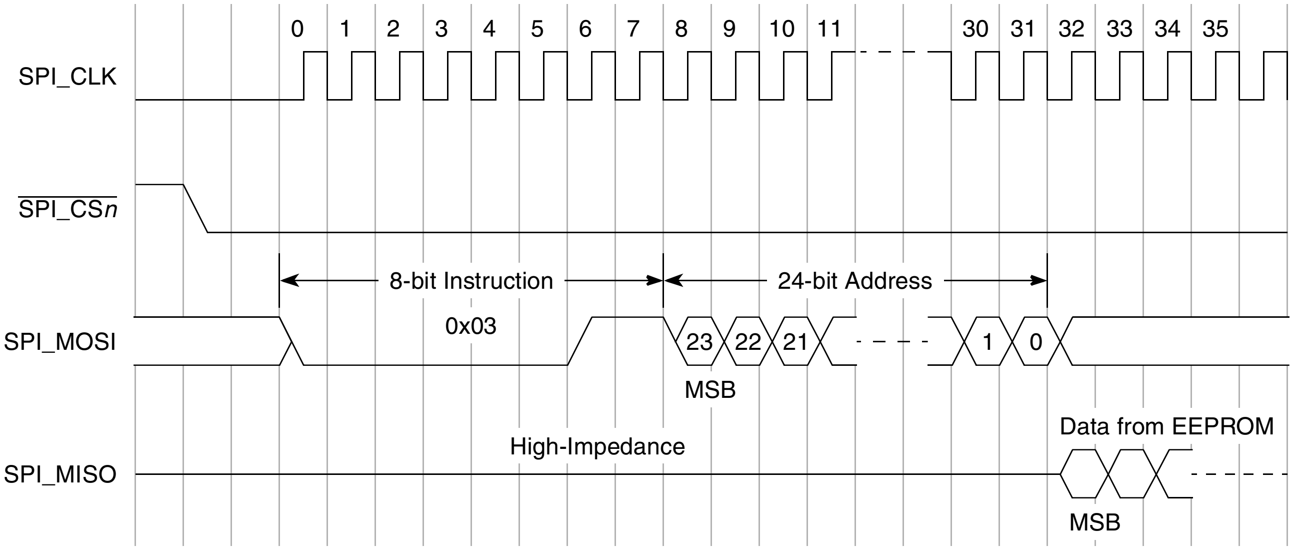


**Figure 4-15. Read Instruction Timing Diagram (24-bit addressable eSPI memory)**

### 4.5.1.3 Default e500 Addressing During System Boot

During boot from the on-chip ROM (for boot targets of either eSPI or SD/MMC), the user specifies 32-bit addresses for several fields (Target Address for copying the user’s code, and the Execution Starting Address). This section describes how these 32-bit effective addresses are translated into 36-bit real addresses and the associated address translation and mapping.

The L2 cache remains disabled as per its power-on reset state. The e500 Level 1 and Level 2 MMU configuration is left as per defaults, with the exception that the following TLB1 Entry 1 is also created (in addition to the default TLB1 Entry 0 4kByte page at 0x0_FFFF_F*nnnn*):

- V=1 (valid)
- TS=0 (address space 0)
- TID=0x00 (global)
- EPN[32–51]=0x00000
- RPN[32–51]=0x00000
- SIZE[0–3]=1011 (4 Gbyte)
- SX/SR/SW=111 (Full supervisor mode access allowed)

- UX/UR/UW=000 (No user mode access allowed)
- WIMGE=01110 (Cache-inhibited, Memory coherency required, Guarded)
- X0–X1=00
- U0–U3=0
- IPROT=1 (Page is protected from invalidation)

This configuration results in a 32-bit byte address with a 0-bit effective page number. Therefore the 36-bit real address is equal to the 32-bit effective address, with the 4 MSbits of the 36-bit real address equal to 0.

The on-chip ROM code does not setup any Local Access Windows. Access to CCSR address space (and therefore by extension, also access to the on-chip ROM) doesn't require a Local Access Window. It is the user's responsibility to setup a local access window through a Control Word address/data pair for the desired Target Address and Execution Starting Address (which will typically be in either DDR or Local Bus memory space).

Note that any such local access window configured at this time must have the 4 MSbits of the address equal to 0. This is due to the 32-bit addressing enabled by the e500 MMUs as described above.

The user can reconfigure the system in the user code portion based on system requirements.

# Part II
# e500 Core Complex and L2 Cache

This part describes the integration of the e500v2 core in the MPC8536E and the interaction between the core complex and the L2 cache. The following chapters are included:

- Chapter 5, “e500 Core Integration Details,” provides an overview of the e500v2 core processor and how it is implemented in the MPC8536E.
- Chapter 6, “L2 Look-Aside Cache/SRAM,” describes the L2 cache of the MPC8536E. Note that the L2 cache can also be addressed directly as memory-mapped SRAM.

# Chapter 5
# e500 Core Integration Details

This chapter descibes how the core is integrated into the SoC. Section 5.2, "e500 Core Integration and the Core Complex Bus (CCB)," describes hardware aspects of that integration and provides links to chapters that discuss functionality in which core and SoC operations interact. Such topics include reset, power management, interrupt management, and debug.

This chapter also lists SoC-specific details of the core's programming model. For example, the e500 programming model defines the processor version register (PVR) and system version register (SVR), special-purpose registers (SPRs) that respectively identify the version and revision of the core and of the integrated device. These values are provided in Section 5.3, "Summary of Core Integratation Details," and additional links are provided to other chapters that provide a context for a discussion of these registers. The section "Register Model Integration Details" in Table 5-1 describes a few aspects of the core programming model that have SoC-specific behavior that cannot be fully understood by reading the *e500 Reference Manual* alone.

General information about e500 core functionality can be found in the following documention:

- Chapter 5, "e500 Core Integration Details," includes a general summary of e500 core features.
- The *PowerPC™ e500 Core Family Reference Manual* (referrred to here as the *e500 Reference Manual*) provides detailed information about the functions and features of the core, and in particular it describes details of how architecture-defined features are implemented.

  The "e500 Core Complex Bus (CCB) and System Integration" chapter in the *e500 Reference Manual* describes core-to-SoC integration issues from the perspective of the e500 core.
- The *EREF: A Programmer's Reference Manual for Freescale Embedded Processors (Including the e200 and e500 Families)* describes in detail features defined by the Power ISA and Freescale EIS (referred to generically as the architecture). Unless otherwise stated in the *e500 Reference Manual*, e500 features are implemented as the are defined by the architecture and described in the *EREF*.
- How the integrated device implements e500 core features, including specific registers and register fields, is summarized in Section 5.3, "Summary of Core Integratation Details."

## 5.1 e500 Core Overview

Figure 5-1 is a block diagram of the processor core complex that shows how the functional units operate independently and in parallel. Note that this is a conceptual diagram that does not attempt to show how these features are physically implemented.

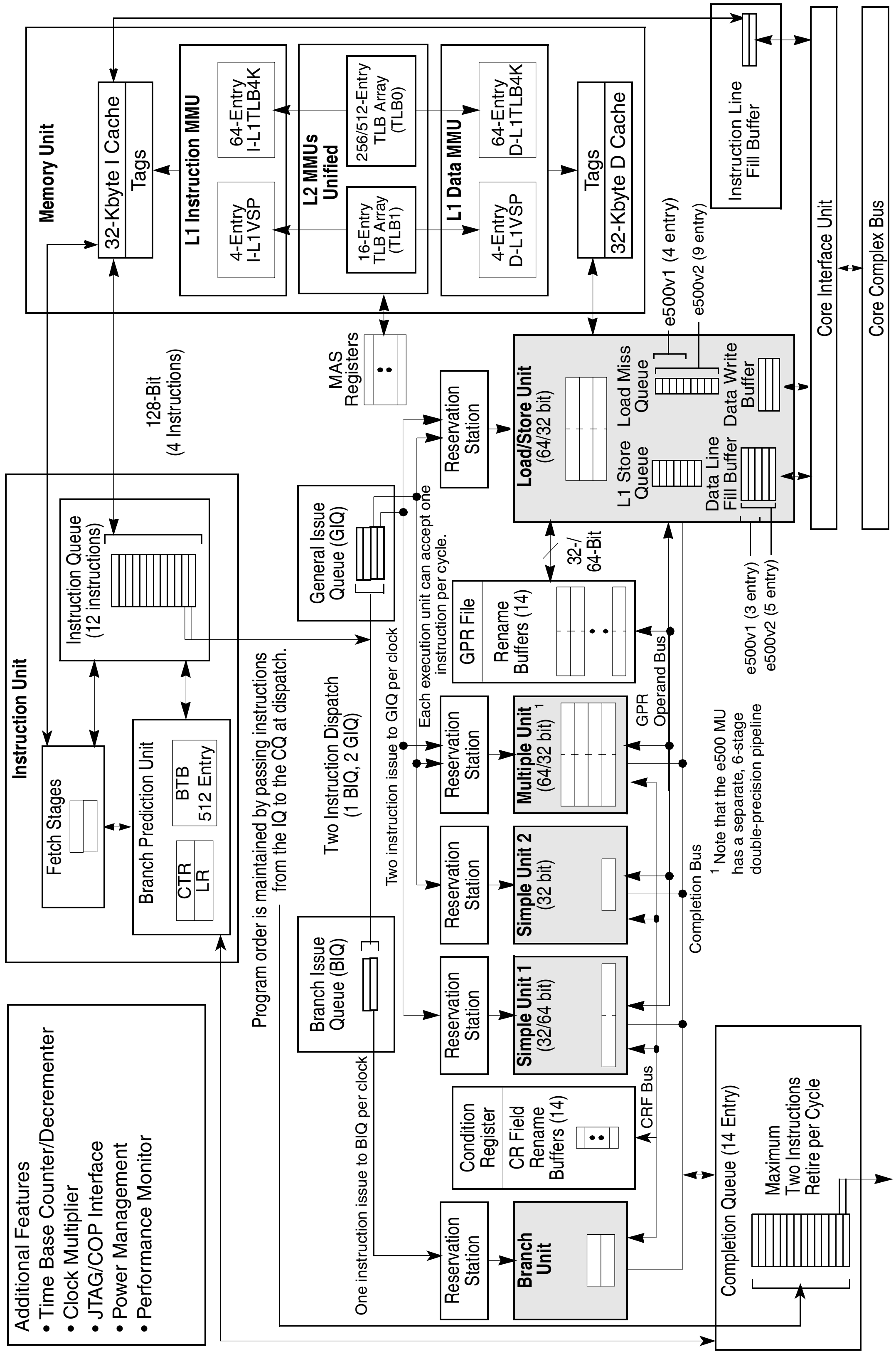


**Figure 5-1. e500 Core Complex Block Diagram**

## 5.2 e500 Core Integration and the Core Complex Bus (CCB)

The CCB is the hardware interface between the core and the SoC, and with a few exceptions, the user cannot access these internal signals directly. Figure 5-2 shows a selection of CCB signals that are discussed in various places in this manual and in the *e500 Reference Manual* because understanding how they work helps the reader understand the functionality of the device. Links to other chapters in this manual are provided in Figure 5-2 and in the text that follows.

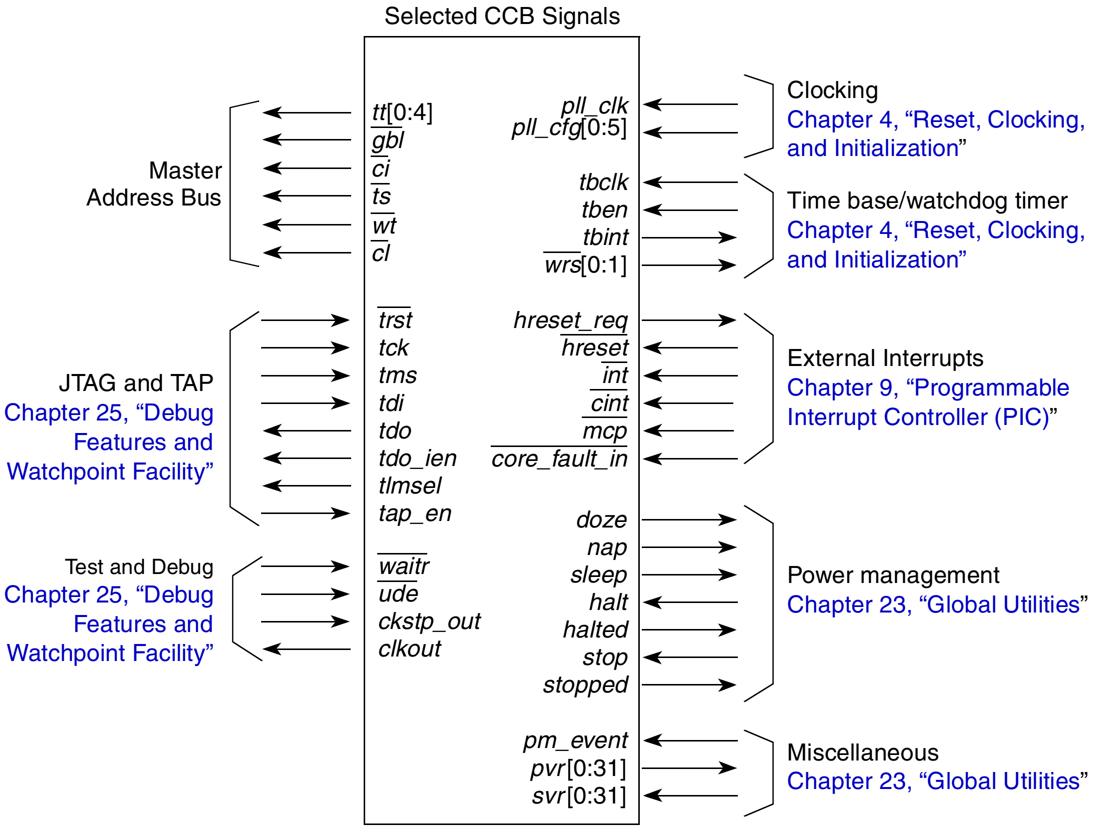


**Figure 5-2. e500 Core Integration**

Aspects of the e500–SoC integration are summarized in the following:

- Reset. The core directs and coordinates device hard and soft resets and the power-on reset (POR) sequence, power-on reset configuration, and initialization. Core integration of the reset signals shown in Figure 5-2 are described Chapter 4, “Reset, Clocking, and Initialization,” and in Chapter 23, “Global Utilities.”
- Clocking and timers. Integration details of the CCB clocking signals are described in Chapter 4, “Reset, Clocking, and Initialization.” Addition details regarding timer configuration are described in Section 5.3, “Summary of Core Integratation Details.”
- Cache and memory-mapped SRAM. The e500 cache implementation interacts with the SoC’s L2 cache. In particular, the core implements a number of instructions that interact with the L2 cache implementation, which are described in the *e500 Reference Manual* and in the *EREF*. Chapter 6, “L2 Look-Aside Cache/SRAM,” describes the SoC’s L2 cache. Figure 5-2 shows the e500 signals that interface with the L2 cache.

- e500 coherency module (ECM). The ECM, described in Chapter 7, “e500 Coherency Module,” facilitates communication between the core, the L2 cache, and the other blocks that comprise the coherent memory domain of the SoC.

  The ECM provides a mechanism for I/O-initiated transactions to snoop the core complex bus (CCB) of the core to maintain coherency across cacheable local memory. It also provides a flexible, easily expandable switch-type structure for core- and I/O-initiated transactions to be routed (dispatched) to target modules on the device. The CCB is described in the “Core Complex Bus (CCB) and System Integration” chapter of the *e500 Reference Manual*.
- Interrupts. The e500 core handles the hardware interrupts that are generated from SoC peripheral logic, typically by error conditions encountered from within blocks on the integrated device. Chapter 9, “Programmable Interrupt Controller (PIC),” describes the programmable interrupt controller, which prioritizes interrupt requests to the core. Figure 5-2 shows the e500 signals that interface with the PIC.
- Power management. The core’s HID0[NAP, DOZE,SLEEP] can be used to assert *nap*, *doze*, and *sleep* output signals to initiate power-saving modes at the integrated device level. Figure 5-2 shows the e500 signals, which interact with the S0C level power management logic described in Chapter 23, “Global Utilities.”
- System debug. The architecture defines many features for software and hardware debug that interact with the SoC. Chapter 25, “Debug Features and Watchpoint Facility,” describes the debug features and watchpoint monitor. Figure 5-2 shows the e500 signals that interface with the debug block. The *e500 Reference Manual* describes how the architecture-defined debug resources are implemented on the e500 core.

The “Core Complex Bus (CCB) and System Integration” chapter of the core document describes the signals that are shown in Figure 5-2 and other aspects of core integration.

## 5.3 Summary of Core Integration Details

Table 5-1 summarizes details of the PowerQUICC III–specific implementation of the e500 core. It is organized into two sections:

- “General Feature Integration Details” summarizes integration-specific details by functionality.
- “Register Model Integration Details” summarizes how integration-specific details are reflected in the SoC’s implementation of the core register model.

**Table 5-1. Differences Between the e500 Core and the PowerQUICC III Core Implementation**

| Feature | PowerQUICC III Implementation |
|---|---|
| **General Feature Integration Details** | |
| Cache protocol | The write-through L2 cache implemented on the SoC does not support MESI cache protocol. |
| Clocking | Internal clock multipliers ranging from 1 to 8 times the bus clock, including integer and half-mode multipliers.The integrated device supports multipliers of 2, 2.5, 3, and 3.5 See the table entry, HID1 Implementation. |

### Table 5-1. Differences Between the e500 Core and the PowerQUICC III Core Implementation (continued)

| Feature | PowerQUICC III Implementation |
|---|---|
| Multiprocessor functionality | Because PowerQUICC III is designed for a uniprocessor environment, the following e500 functionality is not implemented:<br>• The memory coherence bit, M, which is controlled through MAS2[M] and MAS4[MD] has no effect.<br>• HID1[ABE] has meaning only in that it must be set to ensure that cache and TLB management instructions operate properly with respect to the L2 cache.<br>• Dynamic snooping does not occur in power-stopped state (see the note below in the entry for dynamic bus snooping). |
| R1 and R2 data bus parity | R1 and R2 data bus parity are disabled on PowerQUICC III devices. HID1[R1DPE,R2DPE] are reserved. |
| Dynamic bus snooping | The PowerQUICC III devices do not perform dynamic bus snooping as described here. That is, when the e500 core is in core-stopped state (which is the state of the core when the PowerQUICC III device is in either the nap or sleep state), the core is not awakened to perform snoops on global transactions. |
| Device specific definition for TCR[WRC] | PowerQUICC III devices define values for 00, 01, 10, and 11, as described in Register Model Integration Details in this table. |
| SPE and floating-point categories | The SPE (which includes the embedded vector and scalar floating-point instructions) will not be implemented in the next generation of PowerQUICC devices. Freescale Semiconductor strongly recommends that use of these instructions be confined to libraries and device drivers. Customer software that uses these instructions at the assembly level or that uses SPE or floating-point intrinsics will require rewriting for upward compatibility with next generation PowerQUICC devices.<br>The e500v2 core implements SPE double-precision floating-point instructions.<br>Freescale Semiconductor offers a libcfsl_e500 library that uses SPE instructions. Freescale Semiconductor will also provide future libraries to support next generation PowerQUICC devices.<br>Note that in the Power ISA, MSR[SPE] and ESR[SPE] are renamed to MSR[SPV] and ESR[SPV]. |
| **Register Model Integration Details** | |
| PIR value | The PIR value is all zeros on PowerQUICC III devices. |
| PVR value | The PVR reset value is 0x80*nn_nnnn.* See Table 5-2 for specific values.<br>PVR[VERSION] = 0x80*nn*<br>PVR[REVISION] = 0x*nnnn* |
| SVR value | The SVR reset value is 0x80*nn_nnnn.* See Table 5-2 for specific values. |
| TCR (timer control register () | TCR[WRC] is defined more specifically for the implementation of the core in the integrated device.<br>Watchdog timer reset control. This value is written into TSR[WRS] when a watchdog event occurs. WRC may be set by software but cannot be cleared by software, except by a software-induced reset. Once written to a non-zero value, WRC may no longer be altered by software.<br>00 No watchdog timer reset can occur.<br>01 Force processor checkstop on second timeout of watchdog timer<br>10 Assert processor reset output (*core_hreset_req*) on second timeout of watchdog timer<br>11 Reserved |
| HID0 implementation | SEL_TBCLK. Selects time base clock. If this bit is set and the time base is enabled, the time base is based on the TBCLK input, which on the PowerQUICC III devices is RTC. |

**Table 5-1. Differences Between the e500 Core and the PowerQUICC III Core Implementation (continued)**

| Feature | PowerQUICC III Implementation |
|---|---|
| HID1 Implementation | HID1[PLL_CFG] is implemented as two subfields:<br>PLL_MODE (HID0[32–33]):<br>Read-only for integrated devices.<br>11 Fixed value for this device<br>PLL_CFG, (HID0[34–39]): The following clock ratios are supported:<br>0001_00 Ratio of 2:1<br>0001_01 Ratio of 5:2 (2.5:1)<br>0001_10 Ratio of 3:1<br>0001_11 Ratio of 7:2 (3.5:1)<br>NEXEN, R1DPE, R2DPE, MPXTT, MSHARS, SSHAR, ATS, and MID are not implemented.<br>On PowerQUICC III devices, ABE must be set to ensure that cache and TLB management instructions operate properly on the L2 cache. |
| | HID1[RFXE].<br>If RFXE is 0, conditions that cause the assertion of $\overline{core\_fault\_in}$ cannot directly cause the e500 to generate a machine check; however, PowerQUICC III devices must be configured to detect and enable such conditions. The following describes how error bits should be configured:<br>• ECM mapping errors: EEER[LAEE] must be set. See Section 7.2.1.6, “ECM Error Enable Register (EEER).”<br>• L2 multiple-bit ECC errors: L2ERRDIS[MBECCDIS] must be cleared to ensure that error can be detected. L2ERRINTEN[MBECCINTEN] must be set. See Section 6.3.1.4, “L2 Error Registers.”<br>• DDR multiple-bit ECC errors. ERR_DISABLE[MBED] and ERR_INT_EN[MBEE] must be zero and DDR_SDRAM_CFG[ECC_EN] must be one to ensure that an interrupt is generated. See Section 8.4.1, “Register Descriptions.”<br>• PCI. The appropriate parity detect and master-abort bits in ERR_DR must be cleared and the corresponding enable bits in ERR_EN must be set to ensure that an interrupt is generated.<br>• Local bus controller parity errors. LTEDR[PARD] must be cleared and LTEIR[PARI] must be set to ensure that an parity errors can generate an interrupt. See Section 13.3.1.11, “Transfer Error Interrupt Enable Register (LTEIR),” and Section 13.3.1.12, “Transfer Error Attributes Register (LTEATR).” |

## 5.3.1 Processor Version Register (PVR) and System Version Register (SVR)

Table 5-2 matches the revision codes in the processor version register (PVR) with the core revision and the SoC revision; it includes a cross-reference to the section in global utilities that contains the corresponding SVR values. Note that the SVR and PVR can be accessed both as SPRs through the e500 core (see the e500 Reference Manual) and as memory-mapped registers defined by the integrated device. (See “Section 23.4.1.20, “Processor Version Register (PVR),” and Section 23.4.1.21, “System Version Register (SVR),” for further description of the memory-mapped registers.)

**Table 5-2. Device Revision Level Cross-Reference**

| SoC Revision | Core Revision | Processor Version Register (PVR) | System Version Register (SVR) |
|---|---|---|---|
| 1.1 | 3.0 | 0x8021_0030 | 23.4.1.21/23-30 |

# Chapter 6
# L2 Look-Aside Cache/SRAM

This chapter describes the organization of the on-chip L2/SRAM, cache coherency rules, cache line replacement algorithm, cache control instructions, and various cache operations. It also describes the interaction between the L2/SRAM and the e500 core complex.

## 6.1 L2 Cache Overview

The integrated 512-Kbyte L2 cache is organized as 2048 eight-way sets of 32-byte cache lines based on 36-bit physical addresses, as shown in Figure 6-1.

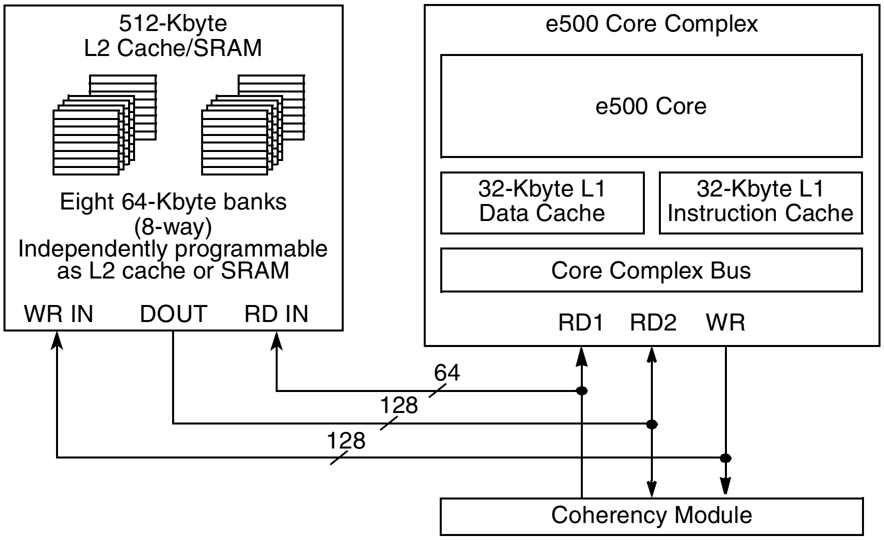


**Figure 6-1. L2 Cache/SRAM Configuration**

The SRAM can be configured with memory-mapped registers as externally accessible memory-mapped SRAM in addition to or instead of cache. The L2 cache can operate in the following modes, described in Section 6.2, “L2 Cache and SRAM Organization”:

- Full cache mode (512-Kbyte cache).
- Full memory-mapped SRAM mode (512-Kbyte SRAM mapped as a single 512-Kbyte block or two 256-Kbyte blocks)
- Partial SRAM and partial cache mode, in which one eighth, one quarter, or one half the total on-chip memory can be allocated to 1 or 2 SRAM regions.

### 6.1.1 L2 Cache and SRAM Features

The L2 cache has the following characteristics:

- Supports 36-bit address space
- Write-through, front-side cache
  - Front-side design provides easier cache access for the I/O masters, such as Ethernet
  - Write-through design is more efficient on the processor bus for front-side caches
- Valid, locked, and stale states (no modified state)
- Two input data buses (64 and 128 bits wide) and one output data bus (128 bits wide)
- All accesses are fully pipelined and non-blocking (allows hits under misses)
- 512-Kbyte array organized as 2048 eight-way sets of 32-byte cache lines
- Eight-way set associativity with a pseudo-LRU (7-bit) replacement algorithm.
  - High level of associativity yields good performance even with many lines locked or used as SRAM regions
- I/O devices can store data into the cache in a process called ‘stashing.’
  - Stashing is indicated for global I/O writes either by a transaction attribute or by a programmable memory range
  - Regions of the cache can be reserved exclusively for stashing to prevent pollution of processor cache regions.
- Processor L2 cache regions are configurable to allocate instructions, data, or both.
  - External masters can force data to be allocated into the cache through programmed memory ranges or special transaction types (stashing).
    - 1, 2, or 4 ways can be configured for stashing only
- Data ECC on 64-bit boundaries (single-error correction, double-error detection)
- Tag arrays use 20 tag bits and 1 tag parity bit per line.
- Multiple cache locking methods supported
  - Individual line locks are set and cleared using e500 cache locking APU instructions—Data Cache Block Touch and Lock Set (**dcbtls**), Data Cache Block Touch for Store and Lock Set (**dcbtstls**), and Instruction Cache Block Touch and Lock Set (**icbtls**).
  - A lock attribute can be attached to write operations.
  - Individual line locks are set and cleared through core-initiated instructions, by external reads or writes, or by accesses to programmed memory ranges defined in L2 cache external write address registers (L2CEWAR*n*).
  - The entire cache can be locked by setting a configuration register appropriately.
- Lock clearing methods
  - Individual locks can be cleared by cache-locking APU instructions—Data Cache Block Lock Clear (**dcblc**) and Instruction Cache Block Lock Clear (**icblc**)—or by a snooped flush unless entire cache is locked.
  - Flash clearing of all instruction and/or data locks is done by writes to configuration registers.
  - An unlock attribute can be attached to a read instruction.

- Error injection modes supported for testing error handling

SRAM features include the following:

- SRAM regions are created by configuring 1, 2, 4 or 8 ways of each set to be reserved for memory-mapped SRAM.
- Regions can reside at any location in the memory map aligned to the SRAM size.
- SRAM memory is byte addressable; for accesses of less than a cache line, ECC is updated using read-modify-write transactions.
- I/O devices access SRAM regions by marking transactions as snoopable (global).

Table 6-1 lists the possible L2 cache/SRAM configurations.

**Table 6-1. Available L2 Cache/SRAM Configurations**

| Cache | Stash-only Region | SRAM Region 1 | SRAM Region 2 |
|---|---|---|---|
| 512 Kbytes (8 ways) | — | — | — |
| 448 Kbytes (7 ways) | — | 64 Kbytes | — |
| | 64 Kbytes | — | — |
| 384 Kbytes (6 ways) | — | 128 Kbytes | — |
| | | 64 Kbytes | 64 Kbytes |
| | 64 Kbytes | 64 Kbytes | — |
| | 128 Kbytes | — | — |
| 320 Kbytes (5 ways) | 64 Kbytes | 128 Kbytes | — |
| | | 64 Kbytes | 64 Kbytes |
| | 128 Kbytes | 64 Kbytes | — |
| 256 Kbytes (4 ways) | — | 256 Kbytes | — |
| | | 128 Kbytes | 128 Kbytes |
| | 128 Kbytes | 128 Kbytes | — |
| | | 64 Kbytes | 64 Kbytes |
| | 256 Kbytes | — | — |
| 192 Kbytes (3 ways) | 64 Kbytes | 256 Kbytes | — |
| | | 128 Kbytes | 128 Kbytes |
| | 256 Kbytes | 64 Kbytes | — |
| 128 Kbytes (2 ways) | 128 Kbytes | 256 Kbytes | — |
| | | 128 Kbytes | 128 Kbytes |
| | 256 Kbytes | 128 Kbytes | — |
| | | 64 Kbytes | 64 Kbytes |
| — | — | 512 Kbytes | — |
| | | 256 Kbytes | 256 Kbytes |
| | 256 Kbytes | 256 Kbytes | — |
| | | 128 Kbytes | 128 Kbytes |

## 6.2 L2 Cache and SRAM Organization

The on-chip memory array has eight banks, each containing 256 sets of eight cache blocks (or ‘ways’), as shown in Figure 6-2. Each block consists of 32 bytes of data and a tag.

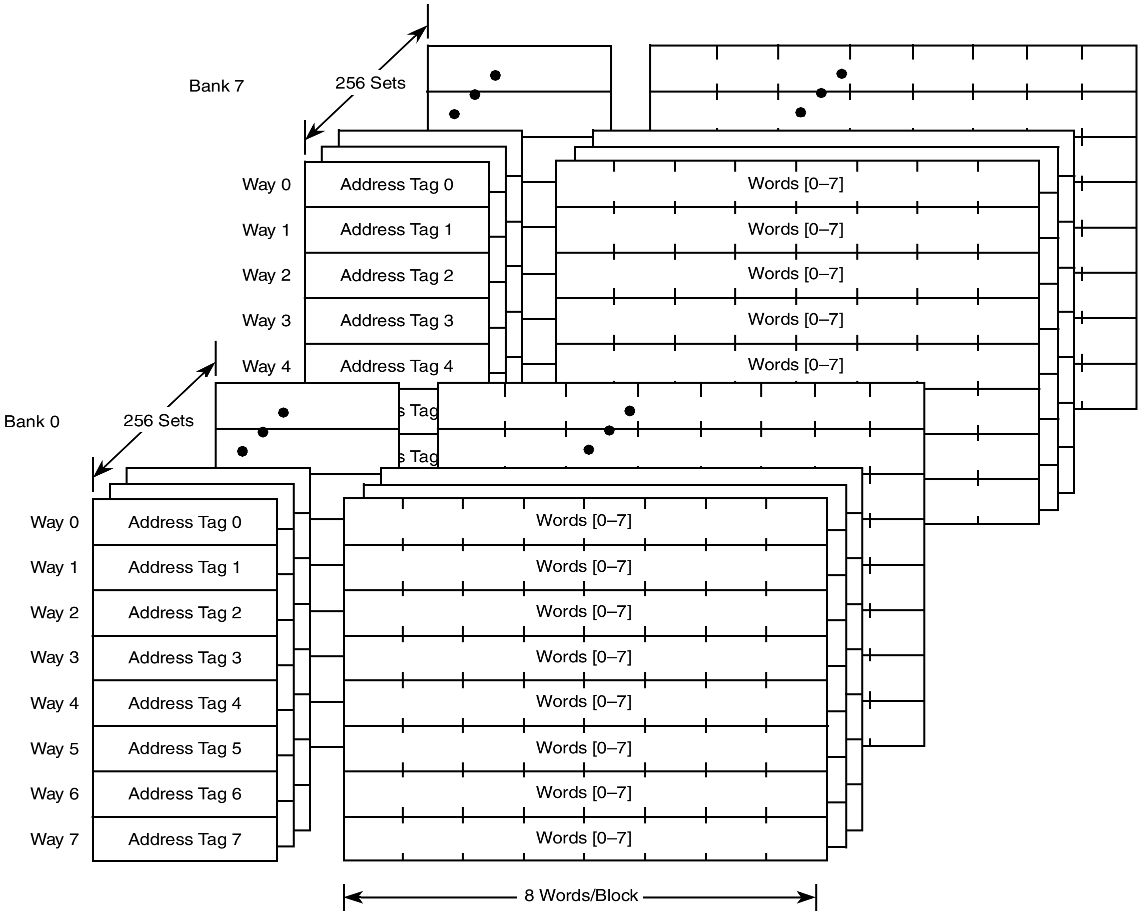


**Figure 6-2. Cache Organization**

### 6.2.1 Accessing the On-Chip Array as an L2 Cache

Figure 6-3 shows how physical address bits are used to access the L2 cache.

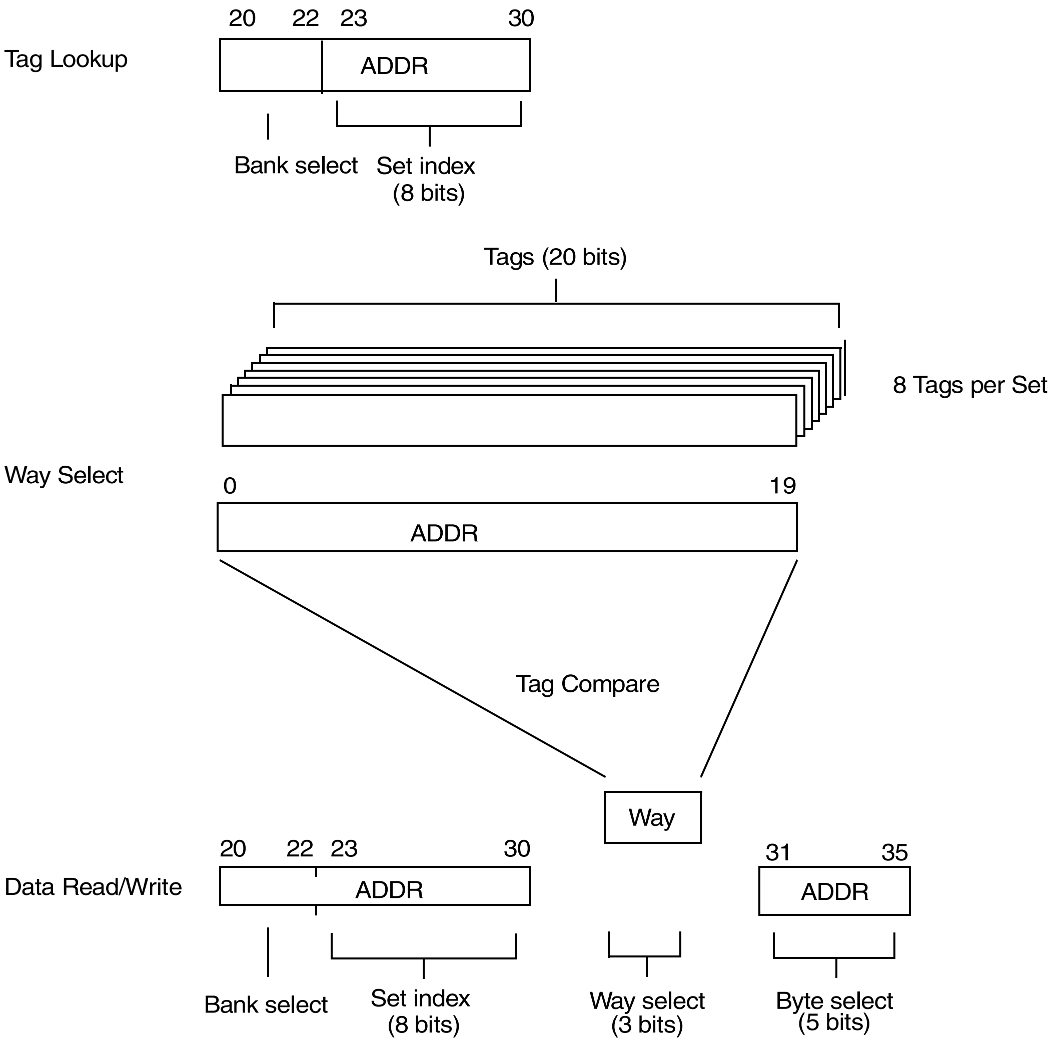


**Figure 6-3. Physical Address Usage for L2 Cache Accesses**

Physical address bits 20–30 identify the bank and set of the tag and data. Physical address bits 0–19 are compared against the tags of all eight ways. A match of a valid tag selects a 32-byte block of data (or way) within the set. Physical address bits 31–35 identify the byte or bytes of data within the block.

### 6.2.2 Accessing the On-Chip Array as an SRAM

When all or part of the array is dedicated to memory mapped SRAM, individual ways of each set are reserved for that purpose. SRAM accesses use physical address bits 17–19 in conjunction with the SRAM mode to select a way of the indexed set.

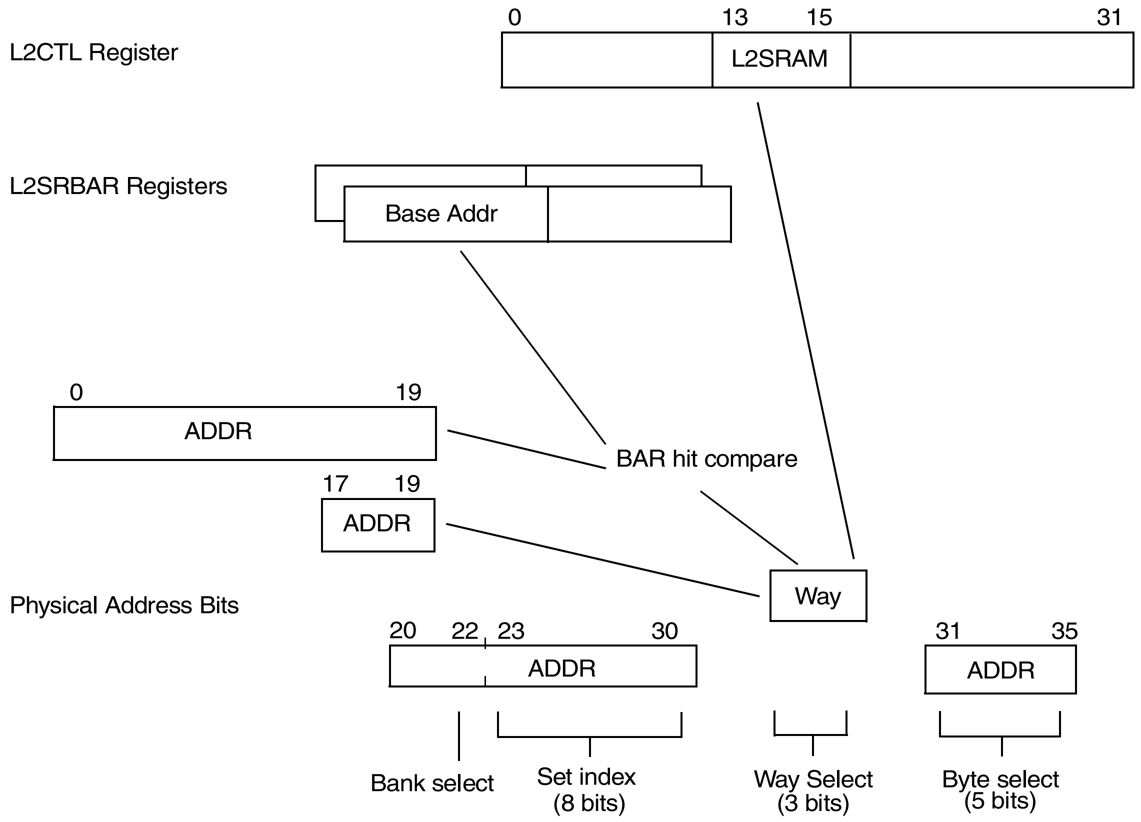


**Figure 6-4. Physical Address Usage for SRAM Accesses**

The mapping of address bits and SRAM mode to a way select is shown below in Table 6-2. SRAM size is reflected in L2CTL[L2SIZ].

**Table 6-2. Way Selection for SRAM Accesses**

| Description | L2SRAM | BAR 0 Hit | BAR 1 Hit | Addr[17–19] | Way Select |
|---|---|---|---|---|---|
| No SRAM | 000 | — | — | — | — |
| Entire Array is SRAM<br>(single 512-KB SRAM if L2SIZ=512 KB) | 001 | 1 | 0 | 000 | 0 |
| | | | | 001 | 1 |
| | | | | 010 | 2 |
| | | | | 011 | 3 |
| | | | | 100 | 4 |
| | | | | 101 | 5 |
| | | | | 110 | 6 |
| | | | | 111 | 7 |

Table 6-2. Way Selection for SRAM Accesses (continued)

| Description | L2SRAM | BAR 0 Hit | BAR 1 Hit | Addr[17–19] | Way Select |
|---|---|---|---|---|---|
| One half of array is an SRAM (single 256-KB SRAM if L2SIZ=512 KB) | 010 | 1 | 0 | x00 | 0 |
| | | | | x01 | 1 |
| | | | | x10 | 2 |
| | | | | x11 | 3 |
| Both halves of array are SRAM (two 256-KB SRAM if L2SIZ=512 KB) | 011 | 1 | 0 | x00 | 0 |
| | | | | x01 | 1 |
| | | | | x10 | 2 |
| | | | | x11 | 3 |
| | | 0 | 1 | x00 | 4 |
| | | | | x01 | 5 |
| | | | | x10 | 6 |
| | | | | x11 | 7 |
| One quarter of the array is SRAM (single 128-KB SRAM if L2SIZ=512 KB) | 100 | 1 | 0 | xx0 | 0 |
| | | | | xx1 | 1 |
| Two quarters of the array are SRAMs (single 128-KB SRAM if L2SIZ=512 KB) | 101 | 1 | 0 | xx0 | 0 |
| | | | | xx1 | 1 |
| | | 0 | 1 | xx0 | 2 |
| | | | | xx1 | 3 |
| One eighth of the array is an SRAM (single 64-KB SRAM if L2SIZ=512 KB) | 110 | 1 | 0 | — | 0 |
| Two eighths of the array are SRAM (single 64-KB SRAM if L2SIZ=512 KB) | 111 | 1 | 0 | — | 0 |
| | | 0 | 1 | — | 1 |

### 6.2.3 Connection of the On-Chip Memory to the System

The e500 core connects to the L2 cache and the system interface through the high-speed core complex bus (CCB). The e500 core and the L2 cache connect to the rest of the integrated device through the e500 coherency module (ECM). Figure 6-5 shows the data connections of the e500 core and L2/SRAM. The e500 core can simultaneously read 128 bits of data from the L2/SRAM, read 64 bits of data from the system interface, and write 128 bits of data to the L2/SRAM and/or system interface.

The L2/SRAM can be accessed by the e500 core or the system interface through the ECM. The L2 cache does not initiate transactions. Figure 6-5 shows the data bus connections of the e500 core and L2/SRAM.

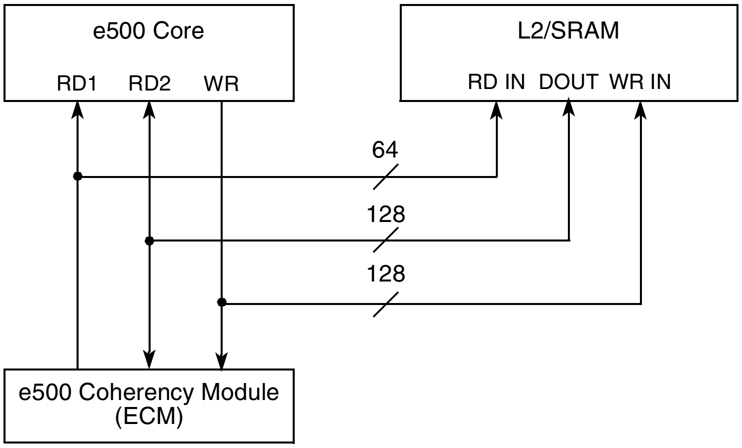


**Figure 6-5. Data Bus Connection of CCB**

Figure 6-6 shows address connections of the e500 core and L2/SRAM.

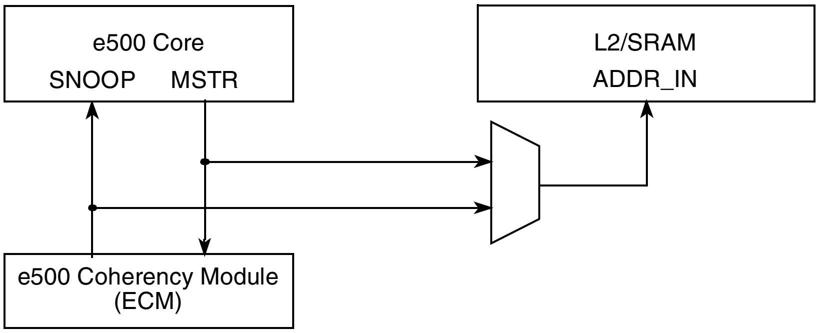


**Figure 6-6. Address Bus Connection of CCB**

In SRAM mode, if a non–cache-line read or write transaction is not preceded by a cache-line write, an ECC error occurs; such a non–cache-line write transaction cannot be allocated in the L2.

## 6.3 Memory Map/Register Definition

Table 6-3 shows the memory map for the L2/SRAM registers.

In this table and in the register figures and field descriptions, the following access definitions apply:

- Reserved fields are always ignored for the purposes of determining access type.
- R/W, R, and W (read/write, read only, and write only) indicate that all the non-reserved fields in a register have the same access type.
- w1c indicates that all of the non-reserved fields in a register are cleared by writing ones to them.
- Mixed indicates a combination of access types.
- Special is used when no other category applies. In this case the register figure and field description table should be read carefully.

Table 6-3. L2/SRAM Memory-Mapped Registers

| Offset | Register | Access | Reset | Section/Page |
|---|---|---|---|---|
| 0x2_0000 | L2CTL—L2 control register | R/W | 0x2000_0000 | 6.3.1.1/6-10 |
| 0x2_0010 | L2CEWAR0—L2 cache external write address register 0 | R/W | 0x0000_0000 | 6.3.1.2.1/6-13 |
| 0x2_0014 | L2CEWAREA0—L2 cache external write address register extended address 0 | R/W | 0x0000_0000 | 6.3.1.2.2/6-14 |
| 0x2_0018 | L2CEWCR0—L2 cache external write control register 0 | R/W | 0x0000_0000 | 6.3.1.2.3/6-14 |
| 0x2_0020 | L2CEWAR1—L2 cache external write address register 1 | R/W | 0x0000_0000 | 6.3.1.2.1/6-13 |
| 0x2_0024 | L2CEWAREA1—L2 cache external write address register extended address 1 | R/W | 0x0000_0000 | 6.3.1.2.2/6-14 |
| 0x2_0028 | L2CEWCR1—L2 cache external write control register 1 | R/W | 0x0000_0000 | 6.3.1.2.3/6-14 |
| 0x2_0030 | L2CEWAR2—L2 cache external write address register 2 | R/W | 0x0000_0000 | 6.3.1.2.1/6-13 |
| 0x2_0034 | L2CEWAREA2—L2 cache external write address register extended address 2 | R/W | 0x0000_0000 | 6.3.1.2.2/6-14 |
| 0x2_0038 | L2CEWCR2—L2 cache external write control register 2 | R/W | 0x0000_0000 | 6.3.1.2.3/6-14 |
| 0x2_0040 | L2CEWAR3—L2 cache external write address register 3 | R/W | 0x0000_0000 | 6.3.1.2.1/6-13 |
| 0x2_0044 | L2CEWAREA3—L2 cache external write address register extended address 3 | R/W | 0x0000_0000 | 6.3.1.2.2/6-14 |
| 0x2_0048 | L2CEWCR3—L2 cache external write control register 3 | R/W | 0x0000_0000 | 6.3.1.2.3/6-14 |
| 0x2_0100 | L2SRBAR0—L2 memory-mapped SRAM base address register 0 | R/W | 0x0000_0000 | 6.3.1.3.1/6-16 |
| 0x2_0104 | L2SRBAREA0—L2 memory-mapped SRAM base address register extended address 0 | R/W | 0x0000_0000 | 6.3.1.3.2/6-17 |
| 0x2_0108 | L2SRBAR1—L2 memory-mapped SRAM base address register 1 | R/W | 0x0000_0000 | 6.3.1.3.1/6-16 |
| 0x2_010C | L2SRBAREA1—L2 memory-mapped SRAM base address register extended address 1 | R/W | 0x0000_0000 | 6.3.1.3.2/6-17 |
| 0x2_0E00 | L2ERRINJHI—L2 error injection mask high register | R/W | 0x0000_0000 | 6.3.1.4.1/6-18 |
| 0x2_0E04 | L2ERRINJLO—L2 error injection mask low register | R/W | 0x0000_0000 | 6.3.1.4.1/6-18 |
| 0x2_0E08 | L2ERRINJCTL—L2 error injection tag/ECC control register | R/W | 0x0000_0000 | 6.3.1.4.1/6-18 |
| 0x2_0E20 | L2CAPTDATAHI—L2 error data high capture register | R | 0x0000_0000 | 6.3.1.4.2/6-20 |
| 0x2_0E24 | L2CAPTDATALO—L2 error data low capture register | R | 0x0000_0000 | 6.3.1.4.2/6-20 |
| 0x2_0E28 | L2CAPTECC—L2 error syndrome register | R | 0x0000_0000 | 6.3.1.4.2/6-20 |
| 0x2_0E40 | L2ERRDET—L2 error detect register | w1c | 0x0000_0000 | 6.3.1.4.2/6-20 |
| 0x2_0E44 | L2ERRDIS—L2 error disable register | R/W | 0x0000_0000 | 6.3.1.4.2/6-20 |
| 0x2_0E48 | L2ERRINTEN—L2 error interrupt enable register | R/W | 0x0000_0000 | 6.3.1.4.2/6-20 |
| 0x2_0E4C | L2ERRATTR—L2 error attributes capture register | R/W | 0x0000_0000 | 6.3.1.4.2/6-20 |
| 0x2_0E50 | L2ERRADDRL—L2 error address capture register low | R | 0x0000_0000 | 6.3.1.4.2/6-20 |
| 0x2_0E54 | L2ERRADDRH—L2 error address capture register high | R | 0x0000_0000 | 6.3.1.4.2/6-20 |
| 0x2_0E58 | L2ERRCTL—L2 error control register | R/W | 0x0000_0000 | 6.3.1.4.2/6-20 |

### 6.3.1 L2/SRAM Register Descriptions

The following sections describe registers that control and configure the L2/SRAM array.

#### 6.3.1.1 L2 Control Register (L2CTL)

The L2 control register (L2CTL), shown in Figure 6-7, controls configuration and operation of the L2/SRAM array. The sequence for modifying L2CTL is as follows:

1. **mbar**
2. **isync**
3. **stw** (WIMG = 01xx) CCSRBAR+0x2_0000
4. **lwz** (WIMG = 01xx) CCSRBAR+0x2_0000
5. **mbar**

Offset 0x2_0000 Access: Read/Write

| | 0 | 1 | 2 | 3 | 4 | | | | 8 | 9 | 10 | 11 | 12 | 13 | | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | L2E | L2I | L2SIZ | | — | | | | | L2DO | L2IO | — | L2INTDIS | L2SRAM | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | | | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | — | | L2LO | L2SLC | — | L2LFR | L2LFRID | | — | | | | L2STASHDIS | — | L2STASHCTL | |
| W | | | | | | | | | | | | | | | | |
| Reset | All zeros | | | | | | | | | | | | | | | |

**Figure 6-7. L2 Control Register (L2CTL)**

Table 6-4 describes L2CTL fields.

**Table 6-4. L2CTL Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | L2E | L2 enable. Used to enable the L2 array (cache or memory-mapped SRAM).<br>0 The L2 SRAM (cache and memory-mapped SRAM) is disabled and is not accessed for reads, snoops, or writes. Setting the L2 flash invalidate bit (L2I) is allowed.<br>1 The L2 SRAM (cache or memory-mapped SRAM) is enabled.<br>Note that L2I can be set regardless of the value of L2E. |
| 1 | L2I | L2 flash invalidate.<br>0 The L2 status and LRU bits are not being cleared.<br>1 Setting L2I invalidates the L2 cache globally by clearing the all the L2 status bits, as well as the LRU algorithm. Memory-mapped SRAM is unaffected.<br>Data to memory-mapped SRAM are unaffected by the flash invalidate. The hardware automatically clears L2I when the invalidate is complete. |
| 2–3 | L2SIZ | L2 SRAM size (read only). Indicates the total available size of on-chip memory array (to be configured as cache or memory-mapped SRAM).<br>00 Reserved<br>01 Reserved<br>10 512 Kbyte<br>11 Reserved |

### Table 6-4. L2CTL Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 4–8 | — | Reserved |
| 9 | L2DO | L2 data-only. Reserved in full memory-mapped SRAM mode. L2DO may be changed while the L2 is enabled or disabled.<br>0 The L2 cache allocates entries for instruction fetches that miss in the L2.<br>1 The L2 cache allocates entries for processor data loads that miss in the L2 and for processor L1 castouts but does not allocate entries for instruction fetches that miss in the L2. Instruction accesses that hit in the L2, data accesses, and accesses from the system (including I/O stash writes) are unaffected.<br>Note that if L2DO and L2IO are both set, no new lines are allocated into the L2 cache for any processor transactions, and processor writes and castouts that hit existing data in the cache invalidate those lines rather than updating them. |
| 10 | L2IO | L2 instruction-only. Reserved in full memory-mapped SRAM mode. Causes the L2 cache to allocate lines for instruction cache transactions only. L2IO may be changed while the L2 is enabled or disabled.<br>0 The L2 cache entries are allocated for data loads that miss in the L2 and for processor L1 castouts.<br>1 The L2 cache allocates entries for instruction fetch misses, but does not allocate entries for processor data transactions. Data accesses that hit in the L2, instruction accesses, and accesses from the system (including I/O stash writes) are unaffected.<br>Note that if L2DO and L2IO are both set, no new lines are allocated into the L2 cache for any processor transactions, and processor writes and castouts that hit existing data in the cache invalidate those lines rather than updating them. |
| 11 | — | Reserved |
| 12 | L2INTDIS | Cache read intervention disable. Reserved for full memory-mapped SRAM mode. Used to disable cache read intervention. May be changed while the L2 is enabled or disabled.<br>0 Cache intervention is enabled. The ECM ensures that if a data read from another device hits in the L2 cache, it is serviced from the L2 cache.<br>1 Cache intervention is disabled |
| 13–15 | L2SRAM | L2 SRAM configuration. Determines the L2 cache/memory-mapped SRAM allocation of the on-chip memory array. SRAM size depends on the value of L2SIZ. Since L2SIZ is 512 Kbytes, SRAM can have sizes from 64 Kbytes to 512 Kbytes.<br>000 No SRAM. Entire array is cache.<br>001 Entire array is a single SRAM (512-Kbyte SRAM for L2SIZ = 512 Kbytes)<br>010 One half of the array is an SRAM (256-Kbyte SRAM for L2SIZ = 512 Kbytes)<br>011 Both halves of the array are SRAMs (two 256-Kbyte SRAMs for L2SIZ = 512 Kbytes)<br>100 One quarter of the array is an SRAM (one 128-Kbyte SRAM for L2SIZ = 512 Kbytes)<br>101 Two quarters of the array are SRAMs (two 128-Kbyte SRAMs for L2SIZ = 512 Kbytes)<br>110 One eighth of the array is an SRAM (one 64-Kbyte SRAM for L2SIZ = 512 Kbytes)<br>111 Two eighths of the array are SRAMs (two 64-Kbyte SRAMs for L2SIZ = 512 Kbytes)<br>For one SRAM region L2SRBAR0 is used and for two SRAM regions L2SRBAR0 and L2SRABAR1 are used. Regions of the array that are not allocated to SRAMs are used as cache memory.<br>To change these bits, the L2 must be disabled (L2CTL[L2E] = 0).<br>Note that when setting L2SRAM after cache has been enabled, L2I should be set as well. The fields can be set simultaneously, and this step is not needed if SRAM size is getting smaller. |
| 16–17 | — | Reserved |

Table 6-4. L2CTL Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 18 | L2LO | L2 cache lock overflow. Reserved in full memory-mapped SRAM mode. This sticky bit is set if an overlock condition is detected in the L2 cache. A lock overflow is triggered either by executing instruction or data cache block touch and lock set instructions or by performing L2 cache external writes with lock set. If all ways are locked and an attempt to stash is made, the stash is not allocated.<br>0 The L2 cache did not encounter a lock overflow. L2LO is cleared only by software.<br>1 The L2 cache encountered a lock overflow condition. |
| 19 | L2SLC | L2 snoop lock clear. This sticky bit is set if a snoop invalidated a locked data cache line. Note that the lock bit for that line is cleared whenever the line is invalidated. L2SLC is reserved in full memory-mapped SRAM mode.<br>0 A snoop did not invalidate a locked L2 cache line. L2SLC is cleared only by software.<br>1 The L2 cache encountered a snoop that invalidated a locked line. |
| 20 | — | Reserved |
| 21 | L2LFR | L2 cache lock bits flash reset. The L2 cache must be enabled (L2CTL[L2E] = 1) for reset to occur. This field is reserved in full memory-mapped SRAM mode.<br>0 The L2 cache lock bits are not cleared or the clear operation completed.<br>1 A reset operation is issued that clears each L2 cache line’s lock bits. Depending on the L2LFRID value, data or instruction locks, or both, can be reset. Cache access is blocked during this time. After L2LFR is set, the L2 cache unit automatically clears L2LFR when the reset operation is complete (if L2CTL[L2E] is set). |
| 22–23 | L2LFRID | L2 cache lock bits flash reset select instruction or data. Indicates whether data or instruction lock bits or both are reset.<br>00 Not used<br>01 Reset data locks if L2LFR = 1.<br>10 Reset instruction locks if L2LFR = 1.<br>11 Reset both data and instruction locks if L2LFR = 1. |
| 24–27 | — | Reserved |
| 28 | L2STASHDIS | L2 stash allocate disable. Disables allocation of lines for stashing.<br>0 The L2 cache allocate lines for global writes that hit in a stash range or that have the stashing attribute set.<br>1 The L2 does not allocate lines for stashed writes.<br>**Note:** This bit does NOT affect the updating of lines that are already resident in the cache and have the stash attribute set or hit a stash range. Such lines are updated even if this bit is set.To change this bit, the L2 must be disabled (L2CTL[L2E] = 0). |

**Table 6-4. L2CTL Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 29 | — | Reserved |
| 30–31 | L2STASHCTL | L2 stash configuration. This field reserves regions of the cache for stash-only operation. That is, blocks of each cache set are reserved so that they can only be allocated for stash data. If such a region is created, processor reads and writes are not allocated into this region; it can only be populated by stash writes. Similarly, stash writes are only allocated into this region. This prevents processor and stashed I/O data from polluting one another.<br>00 No stash-only region. Stashed writes are allocated across the entire cache and can evict processor data and can be evicted by processor data.<br>01 One half of the array is a stash-only cache (way4, way5, way6 & way7 of each set)<br>10 One quarter of the array is a stash-only cache (way6 & way7 of each set)<br>11 One eighth of the array is a stash-only cache (way7 of each set)<br>Like L2SRAM configuration, stash-only regions subtract from the amount of the on-chip memory that is available to the processor as cache. If the L2SRAM configuration uses the entire on-chip memory array as SRAM, then no stash-only region can be created.<br>To change these bits, the L2 must be disabled (L2CTL[L2E] = 0). This field has no effect if the L2STASHDIS bit is set. |

### 6.3.1.2 L2 Cache External Write Registers

The device supports allocating and locking L2 cache lines from external agents such as PCI. This functionality is called stashing. Four sets of registers are provided to support this feature; each set has three registers that specify a programmed memory range that can be locked with a snoop write transaction. All three registers in a set must be configured in order to use an external write address.

These registers are the L2 cache external write address registers 0–3 (L2CEWAR*n*), the L2 cache external write address registers extended address 0–3 (L2CEWAREA*n*), and the L2 cache external write control registers 0–3 (L2CEWCR*n*). L2CEWAR*n* contain the lower 24 bits of the external write base address and L2CEWAREA*n* contain the upper 4 bits. The base address specified in the address registers must be naturally aligned to the window size in the corresponding control register.

Further details on the locations and fields of these registers are given in the following sections.

#### 6.3.1.2.1 L2 Cache External Write Address Registers 0–3 (L2CEWAR*n*)

The L2CEWAR*n* registers contain the lower 24 bits of the 28-bit L2 cache external write base address. Each of these registers has identical fields, as shown in Figure 6-8.

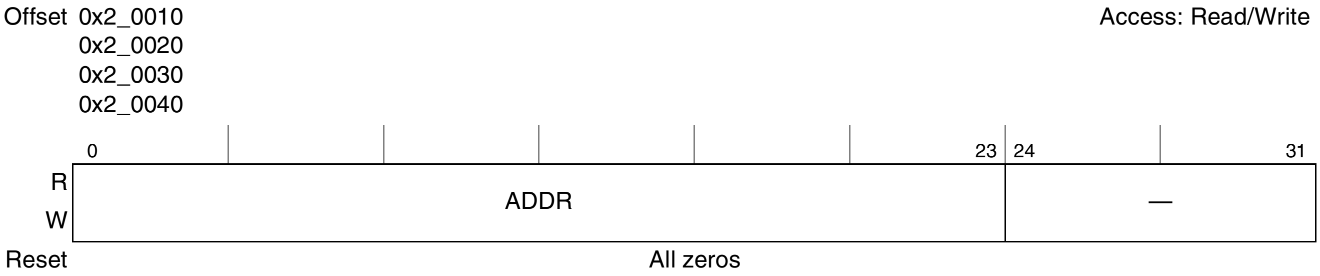


**Figure 6-8. Cache External Write Address Registers (L2CEWAR*n*)**

Table 6-5 describes L2CEWAR*n* fields.

**Table 6-5. L2CEWAR*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–23 | ADDR | Contains the lower 24 bits of the 28-bit L2 cache external write base address. Note that the upper 4 bits of the base address are in L2CEWAREA*n*[ADDR]. |
| 24–31 | — | Reserved |

### 6.3.1.2.2 L2 Cache External Write Address Registers Extended Address 0–3 (L2CEWAREA*n*)

The L2 cache external write address registers extended address (L2CEWAREA*n*), shown in Figure 6-9, contain the upper 4 bits of the 28-bit L2 cache external write base address.

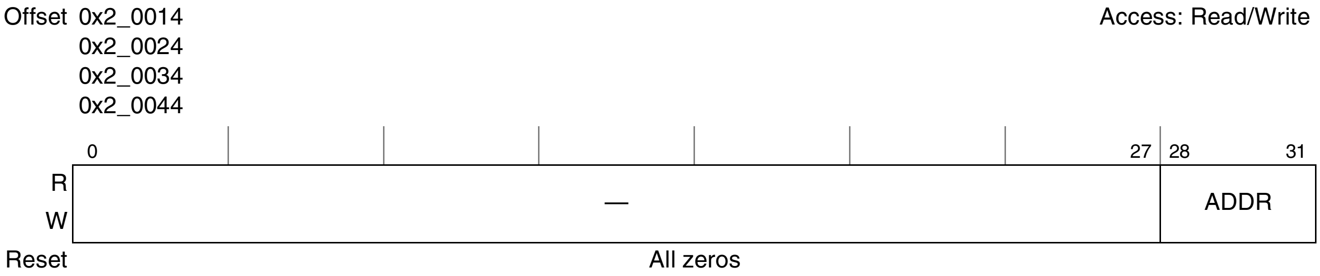


**Figure 6-9. Cache External Write Address Registers Extended Address (L2CEWAREA*n*)**

Table 6-6 describes the fields of L2CEWAREA*n*.

**Table 6-6. L2CEWAREA*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–27 | — | Reserved |
| 28–31 | ADDR | Contains the upper 4 bits of the L2 cache external write base address. Note that the rest of the base address is in L2CEWAR*n*[ADDR]. |

### 6.3.1.2.3 L2 Cache External Write Control Registers 0–3 (L2CEWCR*n*)

The L2CEWAR*n*/L2CEWAREA*n* address registers work with the L2 cache external write control registers 0–3 (L2CEWCR*n*), shown in Figure 6-10, to control cache external write functionality.

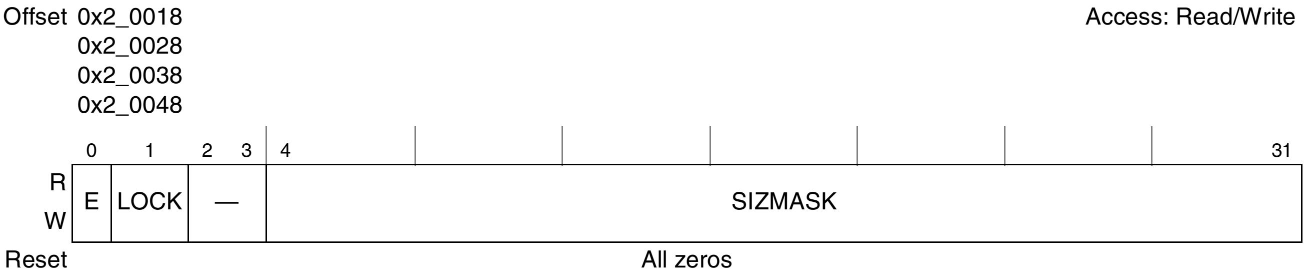


**Figure 6-10. Cache External Write Control Registers (L2CEWCR0–L2CEWCR3)**

The L2CEWCR*n* registers contain identical fields, which are described in Table 6-7.

**Table 6-7. L2CEWCR*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | E | External write enable. An external write matching the address window defined by L2CEWAR*n*/L2CEWAREA*n*/L2CEWCR*n* is allocated or updated in the L2 cache.<br>0 External writes for the L2CEWAR*n*/L2CEWAREA*n*/L2CEWCR*n* set are disabled.<br>1 External writes are enabled for the L2CEWAR*n*/L2CEWAREA*n*/L2CEWCR*n* set. |
| 1 | LOCK | Lock lines in the targeted cache. An external write matching the address window defined by L2CEWAR*n*/L2CEWAREA*n*/L2CEWCR*n* is locked in the L2 cache when it is allocated or updated.<br>0 The locked bit is not set when a line is allocated unless explicitly specified by transaction attributes.<br>1 Cache lines are allocated as locked. A hit to a valid, unlocked line sets the lock. |
| 2–3 | — | Reserved |
| 4–31 | SIZMASK | Mask size. Defines the size of the naturally aligned address region for cache external writes. The address region must be aligned to a boundary that is a multiple of the mask size. Any value not listed below is illegal and produces boundedly undefined results.<br>1111 1111 1111 1111 1111 1111 1111 256 bytes<br>1111 1111 1111 1111 1111 1111 1110 512 bytes<br>1111 1111 1111 1111 1111 1111 1100 1 Kbyte<br>1111 1111 1111 1111 1111 1111 1000 2 Kbytes<br>1111 1111 1111 1111 1111 1111 0000 4 Kbytes<br>1111 1111 1111 1111 1111 1110 0000 8 Kbytes<br>1111 1111 1111 1111 1111 1100 0000 16 Kbytes<br>1111 1111 1111 1111 1111 1000 0000 32 Kbytes<br>1111 1111 1111 1111 1111 0000 0000 64 Kbytes<br>1111 1111 1111 1111 1110 0000 0000 128 Kbytes<br>1111 1111 1111 1111 1100 0000 0000 256 Kbytes<br>1111 1111 1111 1111 1000 0000 0000 512 Kbytes<br>1111 1111 1111 1111 0000 0000 0000 1 Mbyte<br>1111 1111 1111 1110 0000 0000 0000 2 Mbytes<br>1111 1111 1111 1100 0000 0000 0000 4 Mbytes<br>1111 1111 1111 1000 0000 0000 0000 8 Mbytes<br>1111 1111 1111 0000 0000 0000 0000 16 Mbytes<br>1111 1111 1110 0000 0000 0000 0000 32 Mbytes<br>1111 1111 1100 0000 0000 0000 0000 64 Mbytes<br>1111 1111 1000 0000 0000 0000 0000 128 Mbytes<br>1111 1111 0000 0000 0000 0000 0000 256 Mbytes<br>1111 1110 0000 0000 0000 0000 0000 512 Mbytes<br>1111 1100 0000 0000 0000 0000 0000 1 Gbyte<br>1111 1000 0000 0000 0000 0000 0000 2 Gbytes<br>1111 0000 0000 0000 0000 0000 0000 4 Gbytes<br>1110 0000 0000 0000 0000 0000 0000 8 Gbytes<br>1100 0000 0000 0000 0000 0000 0000 16 Gbytes<br>1000 0000 0000 0000 0000 0000 0000 32 Gbytes<br>0000 0000 0000 0000 0000 0000 0000 64 Gbytes |

### 6.3.1.3 L2 Memory-Mapped SRAM Registers

The registers described in this section, the L2 memory-mapped SRAM base address registers 0–1 (L2SRBAR*n*) and the L2 memory-mapped SRAM base address registers extended address 0–1 (L2SRBAREA*n*), control the memory-mapped SRAM mode functionality. Together, these two pairs of registers define memory blocks that can be mapped into the L2 cache.

Specified SRAM base addresses must be aligned to the size of the SRAM region. If L2CTL[L2SRAM] specifies one memory-mapped SRAM block, its base address must be written to the pair L2SRBAR0 and L2SRBAREA0; if it specifies two memory-mapped SRAM blocks, L2SRBAR0 and L2SRBAREA0 are used for the first SRAM block and L2SRBAR1 and L2SRBAREA1 are used for the second block.

#### 6.3.1.3.1 L2 Memory-Mapped SRAM Base Address Registers 0–1 (L2SRBAR*n*)

The L2 memory-mapped SRAM base address registers (L2SRBAR*n*), shown in Figure 6-11, contain the lower 18 bits of the 22-bit SRAM base address.

Offset 0x2_0100
0x2_0108

Access: Read/Write

| | 0–17 | 18–31 |
|---|---|---|
| R<br>W | ADDR | — |
| Reset | All zeros | |

**Figure 6-11. L2 Memory-Mapped SRAM Base Address Registers (L2SRBAR*n*)**

L2SRBAR bits are described in Table 6-8.

**Table 6-8. L2SRBAR*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–17 | ADDR | Contains the lower 18 bits of the 22-bit L2 memory-mapped SRAM base address; the upper 4 bits are contained in L2SRBAREA*n*[ADDR]. (Note that some of these bits may not be needed, depending on how the L2 cache is partitioned.) The combined base address from L2SRBAREA*n*[ADDR] \|\| L2SRBAR*n*[ADDR] is used as follows:<br>SRAM Partition — Bits Required for SRAM Offset — Bits Used for Actual Base Address<br>64 Kbytes — 16 — 20 (0–19)<br>128 Kbytes — 17 — 19 (0–18)<br>256 Kbytes — 18 — 18 (0–17)<br>512 Kbytes — 19 — 17 (0–16)<br>Unused bits of the base address are masked off by the hardware. |
| 18–31 | — | Reserved |

When enabled, the windows defined in L2SRBAR*n* and L2SRBAREA*n* supersede all other mappings of these addresses for processor and global (snoopable) I/O transactions. Therefore, SRAM windows must never overlap configuration space as defined by CCSRBAR (see Section 4.3.1.1.2, “Configuration, Control, and Status Registers Base Address Register (CCSRBAR).”) Overlapping SRAM and local access windows is discouraged because processor and snoopable I/O transactions would map to the SRAM while non-snooped I/O transactions would be mapped by the local access windows. Only if all accesses to the SRAM address range are snoopable can results be consistent if SRAM and local access windows overlap.

#### 6.3.1.3.2 L2 Memory-Mapped SRAM Base Address Registers Extended Address 0–1 (L2SRBAREA*n*)

The L2 memory-mapped SRAM base address registers extended address (L2SRBAREA*n*), shown in Figure 6-12, contain the upper 4 bits of the L2 cache SRAM base address.

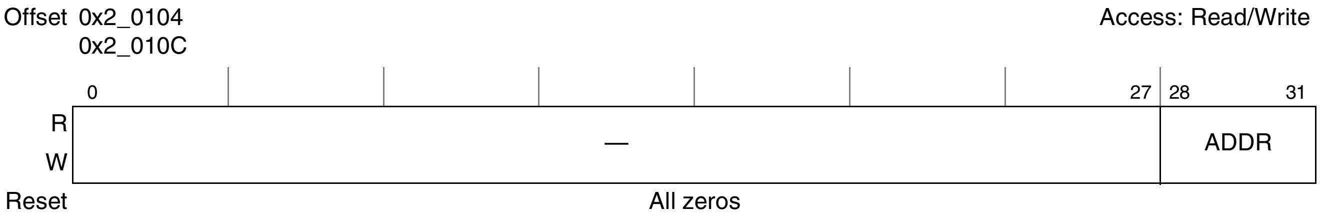


**Figure 6-12. L2 Memory-Mapped SRAM Base Address Registers Extended Address 0–1 (L2SRBAREA*n*)**

Table 6-9 describes the fields of L2SRBAREA*n*.

**Table 6-9. L2SRBAREA*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–27 | — | Reserved |
| 28–31 | ADDR | Contains the upper 4 bits of the L2 cache SRAM base address. Note that the 18 low-order bits of the base address are contained in L2SRBAR*n*[ADDR]. |

### 6.3.1.4 L2 Error Registers

L2 error detection, reporting, and injection allow flexible handling of ECC and parity errors in the L2 data and tag arrays. When the device detects an L2 error, the appropriate bit in the error detect register (L2ERRDET) is set. Error detection is disabled by setting the corresponding bit in the error disable register (L2ERRDIS).

The address and attributes of the first detected error are also saved in the error capture registers (L2ERRADDR, L2ERRATTR, L2CAPTDATAHI, L2CAPTDATALO, and L2CAPTACC). Subsequent errors set error bits in the error detection registers, but information is saved only for the first one. Error reporting (by generating an interrupt) is enabled by setting the corresponding bit in the error interrupt enable register (L2ERRINTEN). Note that the error detect bit is set regardless of the state of the interrupt enable bit. When an error is detected, if error detection is enabled the L2 cache/SRAM always asserts an internal error signal with read data to prevent the L1 caches and architectural registers from being loaded with corrupt data. If error detection is disabled, the detected error bit is not set and no internal signal is asserted.

The L2 error detect register (L2ERRDET) is implemented as a bit-reset type register. Reading from this register occurs normally; however, write operations can clear but not set bits. A bit is cleared whenever the register is written and the data in the corresponding bit location is a 1. For example, to clear bit 6 and not affect any other bits in the register, the value 0x0200_0000 is written to the register.

Note that in SRAM mode, if a non–cache-line read or write transaction is not preceded by a cache-line write, an ECC error occurs; such a non–cache-line write transaction cannot be allocated in the L2.

#### 6.3.1.4.1 Error Injection Registers

The L2 cache includes support for injecting errors into the L2 data, data ECC, or tag. This may be used to test error recovery software by deterministically creating error scenarios.

The preferred method for error injection is to set all data pages to cache-inhibited (MMU TLB entry I = 1) except a scratch page, set L2CTL[L2DO] to prevent allocation of instruction accesses, and invalidate the L2 by setting L2CTL[L2I] = 1. The following code sequence triggers an error, then detects it (A is an address in the scratch page):

```
dcbz A          | allocates the line in the L1 in the modified state
dcbtls_L2 A     | forces the line from the L1 and allocates the line in the L2
lwz A
```

Data or tag errors are injected into the line, according to the error injection settings in L2ERRINJHI, L2ERRINJLO, and L2ERRINJCTL, at allocation. The final load detects and reports the error (if enabled) and allows software to examine the offending data, address, and attributes.

Note that error injection enable bits in L2ERRINJCTL must be cleared by software and the L2 must be invalidated (by setting L2CTL[L2I]) before resuming L2 normal operation. Figure 6-13 shows the L2 error injection mask high register (L2ERRINJHI).

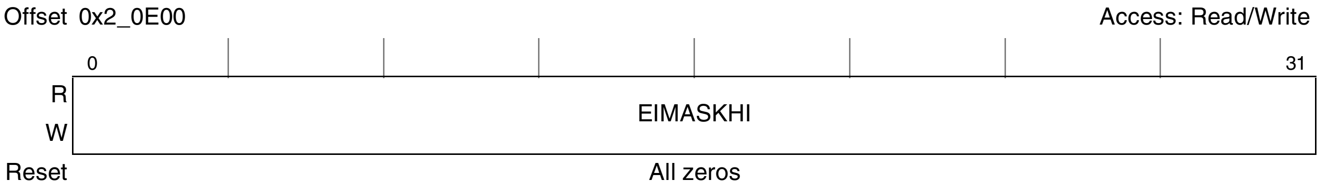


Figure 6-13. L2 Error Injection Mask High Register (L2ERRINJHI)

Table 6-10 describes L2ERRINJHI[EIMASKHI].

Table 6-10. L2ERRINJHI Field Description

| Bits | Name | Description |
|---|---|---|
| 0–31 | EIMASKHI | Error injection mask/high word. A set bit corresponding to a data path bit causes that bit on the data path to be inverted on cache/SRAM writes if L2ERRINJCTL[DERRIEN] = 1. |

Figure 6-14 shows the L2 error injection mask low register (L2ERRINJLO).

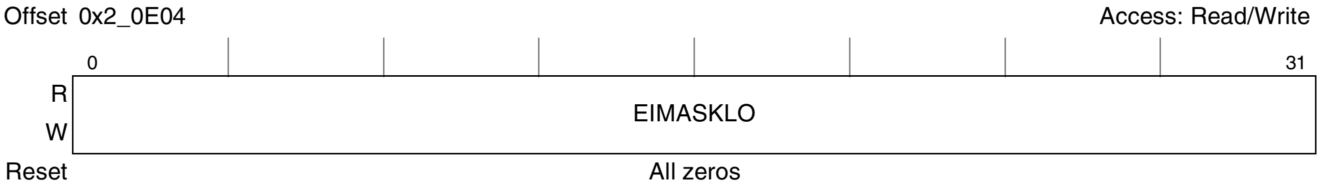


Figure 6-14. L2 Error Injection Mask Low Register (L2ERRINJLO)

Table 6-11 describes L2ERRINJLO[EIMASKLO].

Table 6-11. L2ERRINJLO Field Description

| Bits | Name | Description |
|---|---|---|
| 0–31 | EIMASKLO | Error injection mask/low word. A set bit corresponding to a data path bit causes that bit on the data path to be inverted on SRAM writes if L2ERRINJCTL[DERRIEN] = 1. |

Figure 6-15 shows the L2 error injection mask control register (L2ERRINJCTL).

Offset 0x2_0E08 Access: Read/Write

| | 0 | 14 | 15 | 16 | 21 | 22 | 23 | 24 | 31 |
|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | | TERRIEN | — | | ECCMB | DERRIEN | ECCERRIM | |
| Reset | All zeros | | | | | | | | |

Figure 6-15. L2 Error Injection Mask Control Register (L2ERRINJCTL)

Table 6-12 describes L2ERRINJCTL fields.

Table 6-12. L2ERRINJCTL Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–14 | — | Reserved |
| 15 | TERRIEN | L2 tag array error injection enable<br>0 No tag errors are injected.<br>1 All subsequent entries written to the L2 tag array have the parity bit inverted. |
| 16–21 | — | Reserved |
| 22 | ECCMB | ECC mirror byte enable.<br>0 ECC byte mirroring is disabled<br>1 The most significant data path byte is mirrored onto the ECC byte if DERRIEN = 1. |
| 23 | DERRIEN | L2 data array error injection enable:<br>0 No data errors are injected.<br>1 Subsequent entries written to the L2 data array have data or ECC bits inverted as specified in the data and ECC error injection masks and/or data path byte mirrored onto ECC as specified by ECC mirror byte enable.<br>Note: if both ECC mirror byte and data error injection are enabled, ECC mask error injection is performed on the mirrored ECC. |
| 24–31 | ECCERRIM | Error injection mask for the ECC bits. A set bit corresponding to an ECC bit causes that bit to be inverted on SRAM writes if DERRIEN = 1. |

### 6.3.1.4.2 Error Control and Capture Registers

The error control and capture registers control detection and reporting of tag parity, ECC and L2 configuration errors. L2 configuration errors are illegal combinations of L2 size and block size and are detected when the L2 is enabled (L2CTL[L2E] = 1). Figure 6-16 shows the L2 error capture data high register (L2CAPTDATAHI).

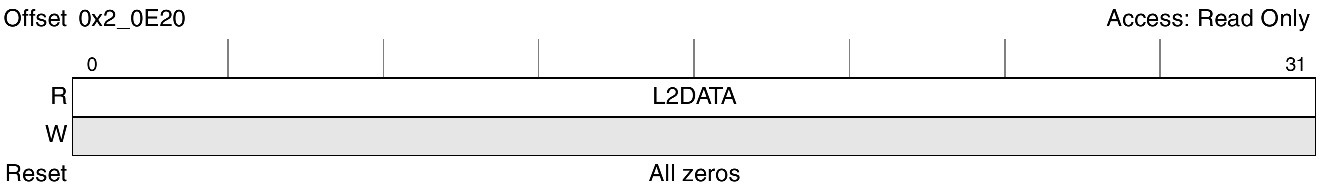


**Figure 6-16. L2 Error Capture Data High Register (L2CAPTDATAHI)**

Table 6-13 describes L2CAPTDATAHI[L2DATA].

**Table 6-13. L2CAPTDATAHI Field Description**

| Bits | Name | Description |
|---|---|---|
| 0–31 | L2DATA | L2 data high word |

Figure 6-17 shows the L2 error capture data low register (L2CAPTDATALO).

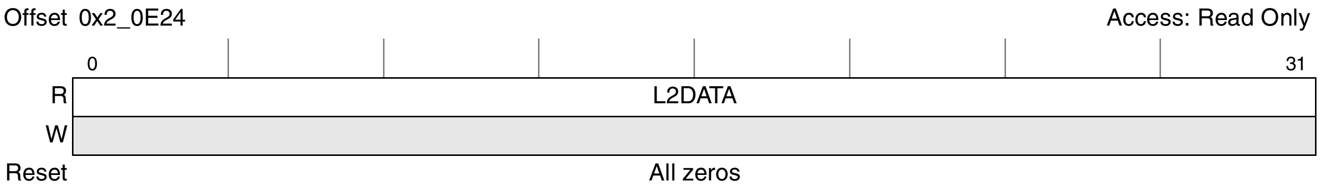


**Figure 6-17. L2 Error Capture Data Low Register (L2CAPTDATALO)**

Table 6-14 describes L2CAPTDATALO[L2DATA].

**Table 6-14. L2CAPTDATALO Field Description**

| Bits | Name | Description |
|---|---|---|
| 0–31 | L2DATA | L2 data low word |

Figure 6-18 shows the L2 error syndrome register (L2CAPTECC).

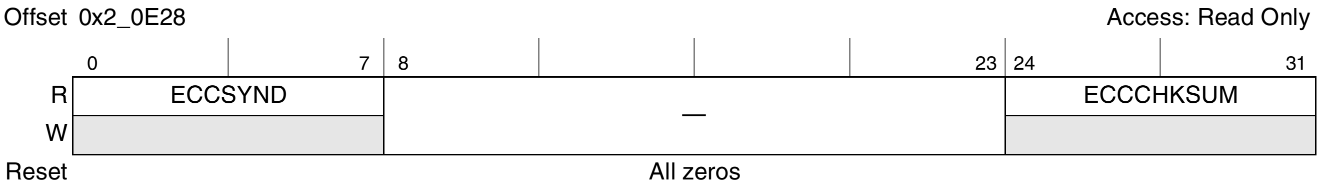


**Figure 6-18. L2 Error Syndrome Register (L2CAPTECC)**

Table 6-15 describes L2CAPTECC fields.

**Table 6-15. L2CAPTECC Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–7 | ECCSYND | The calculated ECC syndrome of the failing double word |
| 8–23 | — | Reserved |
| 24–31 | ECCCHKSUM | The data path ECC of the failing double word |

Figure 6-19 shows the L2 error detect register (L2ERRDET).

Offset 0x2_0E40 Access: W1c

| | 0 | 1–26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|
| R | MULL2ERR | — | TPARERR | MBECCERR | SBECCERR | — | L2CFGDIS |
| W | W1c | | w1c | w1c | w1c | | w1c |
| Reset | All zeros | | | | | | |

**Figure 6-19. L2 Error Detect Register (L2ERRDET)**

Table 6-16 describes L2ERRDET fields.

**Table 6-16. L2ERRDET Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | MULL2ERR | Multiple L2 errors (bit reset, write 1 to clear)<br>0 Multiple L2 errors of the same type were not detected<br>1 Multiple L2 errors of the same type were detected |
| 1–26 | — | Reserved |
| 27 | TPARERR | Tag parity error (bit reset, write 1 to clear)<br>0 Tag parity error was not detected<br>1 Tag parity error was detected<br>Note that if an L2 cache tag parity error occurs on an attempt to write a new line, the L2 cache must be Flash invalidated. L2 functionality is not guaranteed if Flash invalidation is not performed after a tag parity error. |
| 28 | MBECCERR | Multiple-bit ECC error (bit reset, write 1 to clear)<br>0 Multiple-bit ECC errors were not detected<br>1 Multiple-bit ECC errors were detected |
| 29 | SBECCERR | Single-bit ECC error (bit reset, write 1 to clear)<br>0 Single-bit ECC error was not detected<br>1 Single-bit ECC error was detected. |
| 30 | — | Reserved |
| 31 | L2CFGERR | L2 configuration error (bit reset, write 1 to clear)<br>0 L2 configuration errors were not detected<br>1 L2 illegal configuration error detected. Reports inconsistencies between the L2SRAM, L2STASHDIS and L2STASHCTL fields of the L2 control register (L2CTL) |

Figure 6-20 shows the L2 error disable register (L2ERRDIS).

Offset 0x2_0E44 Access: Read/Write

| | 0 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|
| R<br>W | — | TPARDIS | MBECCDIS0 | SBECCDIS | — | L2CFGDIS |
| Reset | All zeros | | | | | |

**Figure 6-20. L2 Error Disable Register (L2ERRDIS)**

Table 6-17 describes L2ERRDIS fields.

**Table 6-17. L2ERRDIS Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–26 | — | Reserved |
| 27 | TPARDIS | Tag parity error disable<br>0 Tag parity error detection enabled<br>1 Tag parity error detection disabled |
| 28 | MBECCDIS | Multiple-bit ECC error disable. Note that uncorrectable read errors may cause the assertion of *core_fault_in*, which causes the core to generate a machine check interrupt, unless it is disabled (by clearing HID1[RFXE]). If RFXE is zero and this error occurs, MBECCDIS must be cleared and L2ERRINTEN[MBECCINTEN] must be set to ensure that an interrupt is generated. For more information, see the reference manual for the e500 core.<br>0 Multiple-bit ECC error detection enabled<br>1 Multiple-bit ECC error detection disabled |
| 29 | SBECCDIS | Single-bit ECC error disable<br>0 Single-bit ECC error detection enabled<br>1 Single-bit ECC error detection disabled |
| 30 | — | Reserved |
| 31 | L2CFGDIS | L2 configuration error disable<br>0 L2 configuration error detection enabled<br>1 L2 configuration error detection disabled |

Figure 6-21 shows the L2 error interrupt enable register (L2ERRINTEN). When an enabled error condition exists, the L2 signals an interrupt to the core through the internal $\overline{int}$ signal.

Offset 0x2_0E48 Access: Read/Write

| | 0 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|
| R<br>W | — | TPARINTEN | MBECCINTEN | SBECCINTEN | — | L2CFGINTEN |
| Reset | All zeros | | | | | |

**Figure 6-21. L2 Error Interrupt Enable Register (L2ERRINTEN)**

Table 6-18 describes L2ERRINTEN fields.

Table 6-18. L2ERRINTEN Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–26 | — | Reserved |
| 27 | TPARINTEN | Tag parity error reporting enable<br>0 Tag parity error reporting disabled<br>1 Tag parity error reporting enabled |
| 28 | MBECCINTEN | Multiple-bit ECC error reporting enable. Note that uncorrectable read errors may cause the assertion of *core_fault_in*, which causes the core to generate a machine check interrupt, unless it is disabled (by clearing HID1[RFXE]). If RFXE is zero and this error occurs, L2ERRDIS[MBECCDIS] must be cleared and MBECCINTEN must be set to ensure that an interrupt is generated. For more information, see the reference manual for the e500 core.<br>0 Multiple-bit ECC error reporting disabled<br>1 Multiple-bit ECC error reporting enabled |
| 29 | SBECCINTEN | Single-bit ECC error reporting enable<br>0 Single-bit ECC error reporting disabled<br>1 Single-bit ECC error reporting enabled |
| 30 | — | Reserved |
| 31 | L2CFGINTEN | L2 configuration error reporting enable<br>0 L2 configuration error reporting disabled<br>1 L2 configuration error reporting enabled |

Figure 6-22 shows the L2 error attributes capture register (L2ERRATTR).

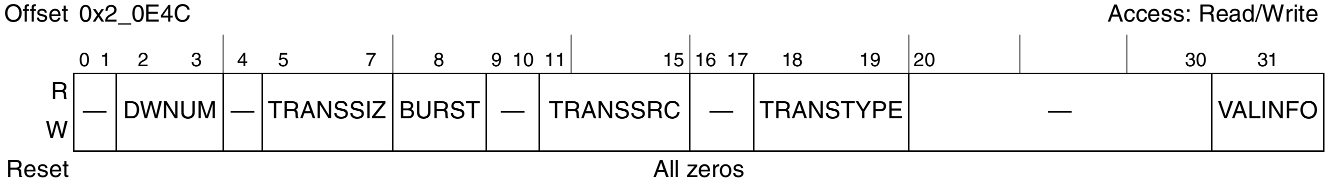


Figure 6-22. L2 Error Attributes Capture Register (L2ERRATTR)

Table 6-19 describes L2ERRATTR fields.

Table 6-19. L2ERRATTR Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–1 | — | Reserved |
| 2–3 | DWNUM | Double-word number of the detected error (data ECC errors only) |
| 4 | — | Reserved |
| 5–7 | TRANSSIZ | Transaction size for detected error<br>Single-beat / Burst / Single-beat / Burst<br>000 8 bytes / Reserved / 100 4 bytes / Reserved<br>001 1 byte / 16 bytes / 101 5 bytes / Reserved<br>010 2 bytes / 32 bytes / 110 6 bytes / Reserved<br>011 3 bytes / Reserved / 111 7 bytes / Reserved |

**Table 6-19. L2ERRATTR Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 8 | BURST | Burst transaction for detected error<br>0 Single-beat (≤ 64 bits) transaction<br>1 Burst transaction |
| 9–10 | — | Reserved |
| 11–15 | TRANSSRC | Transaction source for detected error<br>00000 External (system logic)<br>10000 Processor (instruction)<br>10001 Processor (data) |
| 16–17 | — | Reserved |
| 18–19 | TRANSTYPE | Transaction type for detected error<br>00 Snoop (tag/status read)<br>01 Write<br>10 Read<br>11 Read-modify-write |
| 20–30 | — | Reserved |
| 31 | VALINFO | L2 capture registers valid<br>0 L2 capture registers contain no valid information or no enabled errors were detected.<br>1 L2 capture registers contain information of the first detected error which has reporting enabled. Software must clear this bit to unfreeze error capture so error detection hardware can overwrite the capture address/data/attributes for a newly detected error. |

Figure 6-23 shows the L2 error address capture register low (L2ERRADDRL).

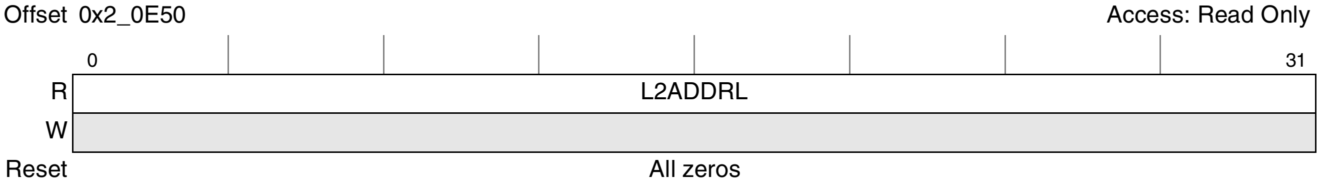


**Figure 6-23. L2 Error Address Capture Register (L2ERRADDRL)**

Table 6-20 describes L2ERRADDRL[L2ADDRL].

**Table 6-20. L2ERRADDRL Field Description**

| Bits | Name | Description |
|---|---|---|
| 0–31 | L2ADDRL | L2 address bits 4–35 corresponding to detected error |

Figure 6-24 shows the L2 error address capture register high (L2ERRADDRH).

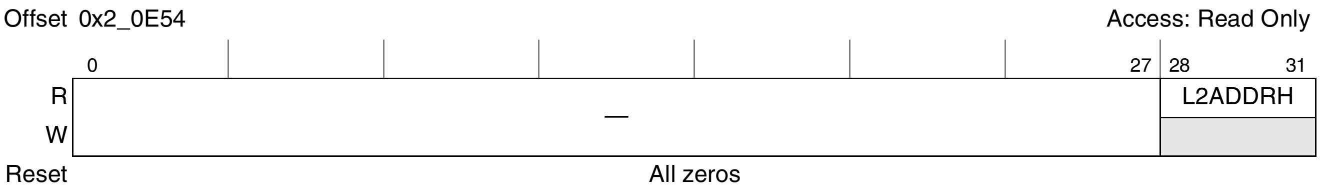


**Figure 6-24. L2 Error Address Capture Register (L2ERRADDRH)**

Table 6-21 describes L2ERRADDRH[L2ADDRH].

**Table 6-21. L2ERRADDRH Field Description**

| Bits | Name | Description |
|---|---|---|
| 0–27 | — | Reserved |
| 28–31 | L2ADDRH | L2 address bits 0–3 corresponding to detected error |

Figure 6-25 shows the L2 error control register (L2ERRCTL).

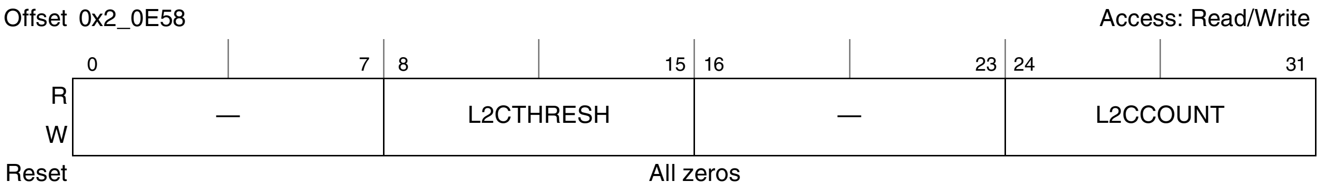


**Figure 6-25. L2 Error Control Register (L2ERRCTL)**

Table 6-22 describes L2ERRCTL fields.

**Table 6-22. L2ERRCTL Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–7 | — | Reserved |
| 8–15 | L2CTHRESH | L2 cache threshold. Threshold value for the number of ECC single-bit errors that are detected before reporting an error condition. |
| 16–23 | — | Reserved |
| 24–31 | L2CCOUNT | L2 count. Counts ECC single-bit errors detected. If L2CCOUNT equals the ECC single-bit error trigger threshold, an error is reported if single-bit error reporting is enabled. |

## 6.4 External Writes to the L2 Cache (Cache Stashing)

Data from an I/O master can be allocated into the L2 cache while simultaneously being written to memory. External (stashed) writes can be performed from any I/O master. For example:

- Ethernet
- PCI/PCI-Express
- DMA

Stashing is controlled either by an attribute from the initiator of a write or by address range registers in the L2 cache. New cache lines are allocated for full-cache-line writes (unless the line is already resident in the cache). Sub-cache-line write data is stashed only if the line is already valid in the cache. For these sub-cache-line writes, a read-modify-write process is used to merge the write data with the valid data already in the cache.

For information on how to initiate cache stashing from an I/O master, see the respective chapters for the I/O masters that support stashing.

For address range based control of stashing, the L2 cache external write address registers 0–3 (L2CEWAR*n*) and the L2 cache external write address registers extended address 0–3 (L2CEWAREA*n*) are used with the L2 cache external write control registers 0–3 (L2CEWCR*n*) to control the cache stashing functionality. Each register set (for example L2CEWAR0, L2CEWAREA0, and L2CEWCR0) specifies a programmed memory range that can be allocated and optionally locked with a global write transaction. The address register must be naturally aligned to the window size in the corresponding control register. For more information, see Section 6.3.1.2, “L2 Cache External Write Registers.”

Note that stashing can occur regardless of whether the L1 cache is enabled or whether the cache-inhibited bit in the MMU is set for the page.

### 6.4.1 Stash-Only Cache Regions

In order to prevent stashed I/O data from polluting processor data in the L2 cache (and vice versa), it is possible to create stash-only regions. This is controlled by the L2STASHCTL field of L2CTL. See Section 6.3.1.1, “L2 Control Register (L2CTL).”

If a stash-only region is created, then that region of the cache is only used for stashed I/O data, and stashed I/O data does not cause the eviction of processor data; they are kept in separate ways of each set. The processor may allocate data into the ways of the cache that are not allocated to SRAM or stash-only memory. Replacement within the stash-only region and the processor region is governed by a pseudo-LRU algorithm modified by masks that allow only applicable ways of a cache set to be considered for replacement.

## 6.5 L2 Cache Timing

Table 6-23 shows the timing of back-to-back loads that miss in the L1 data cache and hit in the L2 cache, assuming the core is running at 2 1/2 times the L2 cache frequency. The L2 returns the 128 bits containing the requested data (critical quad word) first. This data is forwarded to the result register before the full cache line reloads the L1.

Table 6-23. Fastest Read Timing—Hit in L2

| Core Clocks | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| e500 core load 1 | to D-cache | D-cache miss | to CIU | CIU Q | | | | | | | | | | | | to CIU | LSU DLFB | LSU reads command | LSU reads out data | Result bus | | | | | | |
| e500 core load 2 | | to D-cache | D-cache miss | to CIU | CIU Q | | | | | | | | | | | | | | | | to CIU | LSU DLFB | LSU reads command | LSU reads out data | Result bus | |

| CCB clocks | <1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11> |
|---|---|---|---|---|---|---|---|---|---|---|---|
| CCB addr bus load 1 | | BG | TS | | AACK | HIT DATA-COMING | | | | | |
| CCB addr bus load 2 | | | | BG | TS | | AACK | HIT DATA-COMING | | | |
| CCB data bus load 1 | | | | | | | DATA | DATA | | | |
| CCB data bus load 2 | | | | | | | | | DATA | DATA | |

## 6.6 L2 Cache and SRAM Coherency

This section explains the rules of cache and memory-mapped SRAM coherency. The term ‘snoop transaction’ refers to transactions initiated by the system logic or by I/O traffic, as opposed to e500 core-initiated transactions.

### 6.6.1 L2 Cache Coherency Rules

L2 cache coherency rules are as follows:

- The L2 is non-inclusive of the L1—valid L1 lines may be valid or invalid in the L2.
- The L2 cache holds no modified data. Data is in one of four states—invalid, exclusive, exclusive locked, and stale.
- The L2 allocates entries for data cast out or pushed (non-global, non-write-through write with kill) from the L1 caches.
- Lines for e500 core-initiated burst read transactions are allocated as exclusive in the L2.
- The L2 supports I/O devices reading data from valid lines in the L2 cache (data intervention) if L2CTL[L2INTDIS] = 0. An optional unlock attribute causes I/O reads to clear a lock when the read is performed.
- The L2 cache does not respond to cache-inhibited read transactions.
- e500 core-initiated, cache-inhibited store transactions invalidate the line when they hit on a valid L2 line. If the line is locked, it goes to the stale state. For other write transactions the cache-inhibited bit is ignored.
- Non-burst cacheable write transactions from the e500 core (generated by write-through cacheable stores) update a valid L2 cache line through a read-modify-write operation.
- e500 core cast out transactions that hit on a stale line in the L2 cache cause a data update of the line and a change to the valid locked state for that line.
- An e500 core-initiated, cacheable, non-write-through store that misses in the L1 and hits on a line in the L2 invalidates that line in the L2. If the line is marked exclusive locked, the L2 marks the line as stale.
- Transactions that hit a stale L2 cache line that would cause an allocate if they miss cause a data update of the line (when data arrives from memory) and a change to the line’s valid locked state. Data is not supplied by the L2 cache for the read in this case.
- The following transactions kill the data and the respective locks when they hit a valid L2 line:
  - **dcbf**
  - **dcbi**
- The L2 cache supports mixed cache external writes and core-initiated writes to the same addresses if the core-initiated writes are marked coherency-required, caching allowed, not write-through (WIMG = 001x) and the external writes are marked coherency-required, caching-allowed.
- The L2 cache supports writes to the L2 cache from peripheral devices or from I/O controllers through snoop write transactions with addresses that hit in a programmed memory range. Full cache line (32-byte) write transactions update the data for a valid line in the L2 and if the line is not valid in the L2, a line is allocated. Sub-cache line write transactions update the data only for valid L2 cache lines through read-modify-write operations.
- The L2 cache supports burst writes that lock an L2 cache line from peripheral devices or from I/O controllers through write transactions with addresses that hit in a programmed memory range that has the lock attribute set.

- The L2 cache supports burst writes that allocate and/or lock an L2 cache line from peripheral devices or I/O controllers through a write allocate transaction. See the system logic programming model (for example, that of the DMA controller) for details on how to set the transaction type for cache external writes to the L2.

### 6.6.2 Memory-Mapped SRAM Coherency Rules

Memory-mapped SRAM coherency rules are as follows:

- External (non–core-initiated) accesses to memory-mapped SRAM must be marked coherency-required. External accesses to memory-mapped SRAM marked coherency-not-required may cause an address unavailable error.
- Accesses to memory-mapped SRAM are cacheable only in the corresponding e500 L1 caches. External accesses must be marked cache-inhibited or be performed with non-caching transactions.

## 6.7 L2 Cache Locking

The caches can be locked and cleared using the following methods:

- Cache locking methods
  - Individual line locks are set and cleared using instructions defined by the e500 cache locking APU, which is part of the Freescale embedded implementation standards (EIS). These instructions include Data Cache Block Touch and Lock Set (**dcbtls**), Data Cache Block Touch for Store and Lock Set (**dcbtstls**), and Instruction Cache Block Touch and Lock Set (**icbtls**). For detailed information about these instructions, see the *PowerPC e500 Core Reference Manual*.
  - A lock attribute can be attached to write operations.
  - Individual line locks are set and cleared through core-initiated instructions, by external reads or writes, or by accesses to programmed memory ranges defined in L2 cache external write address registers (L2CEWAR*n*).
  - The entire cache can be locked by setting a configuration registers appropriately
- Methods for clearing locks
  - Individual locks can be cleared by cache locking APU instructions (Instruction Cache Block Lock Clear (**icblc**) and Data Cache Block Lock Clear (**dcblc**)) or by snooped flush unless the entire cache is locked.
  - Flash clearing of all instruction and/or data locks can be done by writes to configuration registers.
  - An unlock attribute can be attached to I/O read operations.

### 6.7.1 Locking the Entire L2 Cache

The entire L2 cache can be locked by setting L2CTL[L2DO] = 1 and L2CTL[L2IO] = 1. This has the effect of preventing any further allocation of new lines in the cache by core requests. If there are lines in the cache that are not valid, they cannot be used by core requests until the cache is unlocked. While the cache is locked, read requests are serviced as normal, and snooping continues as normal to maintain coherency. Lines invalidated to satisfy coherency requirements cannot be reallocated by core requests while the cache remains locked. The L2 cache can be unlocked by clearing L2CTL[L2IO] and/or L2CTL[L2DO]. Note that L2CTL[L2DO] and L2CTL[L2IO] have no effect on cache external write allocations or memory-mapped SRAM.

Note that this form of cache locking does not use the lock bits of the cache and cannot be cleared by resetting the cache or lock bits.

### 6.7.2 Locking Programmed Memory Ranges

A programmed memory range can be locked with a snoop write transaction that matches a cache external write address range (specified by L2CEWAR*n*/L2CEWAREA*n* and L2CEWCR*n*). There are no clearing of locks through the programmed address ranges. Locks can be cleared using clear lock instructions, flushes, read-and-clear-lock snoop (RWNITC with clear lock attribute), or flash clear locks.

### 6.7.3 Locking Selected Lines

Individual lines are locked when the L2 receives one of the following burst transactions:

- **icbtls** (CT = 1)—Instruction Cache Block Touch and Lock Set instruction
- **dcbtls** (CT = 1)—Data Cache Block Touch and Lock Set instruction
- **dcbtstls** (CT = 1)—Data Cache Block Touch for Store and Lock Set instruction
- Snoop burst write—If the address hits on a programmed cache external write space with the lock attribute set, or if the write allocate transaction type is used
- Snoop non-burst write—If the address hits on a programmed cache external write space with the lock attribute set

Note that the core complex broadcasts these instructions to the L2 if the CT field in the instruction specifies the L2 cache (CT = 1). When the L2 cache is specified, data is not placed in the L1, only the L2. If the L1 cache is specified (CT = 0), the L2 does not lock the line, and the data is placed in the L1 (and locked).

When the touch lock set L2 instruction (**dcbtls** or **dcbtstls**) hits are modified in the L1 cache, the modified data is allocated into the L2 cache (and written back to main memory) and a data lock is set. The L1 line state transitions to invalid.

Note that if the L2 receives a request to allocate and lock a line, but all lines in the selected way are locked, the requested L2 line is not allocated and the L2 cache lock overflow bit (L2CTL[L2LO]) is set.

Lines invalidated to satisfy coherency requirements cannot be reallocated while the cache remains locked.

### 6.7.4 Clearing Locks on Selected Lines

Individual locks in the L2 are cleared by a lock clear (**icblc** or **dcblc,** CT = 1) instruction. This directs the L2 cache to clear a lock on that line if it hits in the L2 cache. Both data and instruction locks are cleared by the **icblc** and **dcblc** instructions.

Note that the lock on a line is cleared if the line is invalidated by a snooped Flush transaction, and the line in the cache is available for allocation of a new line of instruction or data unless the entire cache is locked.

### 6.7.5 Flash Clearing of Instruction and Data Locks

Locks for instructions and data are recorded separately in the L2 cache, and they can be flash cleared separately by writing the appropriate value to the L2 cache control register (L2CTL[L2LFR] and L2CTL[L2LFRID]). Flash invalidating of the L2 (setting L2CTL[L2I]) clears all locks on both instructions and data.

Note that flash clearing is the only way to clear data locks without clearing instruction locks, or to clear instruction locks without clearing data locks. All instructions and snoop transactions that clear locks clear both data and instruction locks.

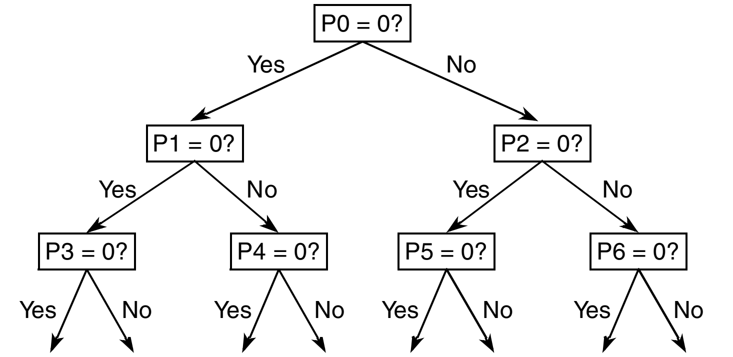
### 6.7.6 Locks with Stale Data

If data is locked in the L2 and either the e500 core performs a cacheable copyback store or a **dcbtst** misses in the L1, the L2 invalidates the line; however, the L2 clears the valid bit for the data, the lock remains, and the line cannot be victimized. If the e500 core casts out modified data or pushes it in response to a non-flush snoop, the L2 updates the data and sets the valid bit again, maintaining the lock and keeping the data in the cache hierarchy.

## 6.8 PLRU L2 Replacement Policy

Line replacement is determined using a pseudo least-recently-used (PLRU) algorithm. There is a valid bit (V0–V7) for each line. To determine the replacement victim (the line to be cast out), there are seven PLRU bits (P0–P6) for each set. PLRU bits are updated every time a new line is allocated and every time an existing line is read by the processor, updated by a write, or invalidated.

Figure 6-26 shows the binary decision tree used to generate the victim line. The eight ways of the L2 cache are labeled W0–W7; the seven PLRU bits are labeled P0–P6.



|  | Allocate W0 | Allocate W1 | Allocate W2 | Allocate W3 | Allocate W4 | Allocate W5 | Allocate W6 | Allocate W7 |
|---|---|---|---|---|---|---|---|---|
| P0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| P1 | 0 | 0 | 1 | 1 | x | x | x | x |
| P2 | x | x | x | x | 0 | 0 | 1 | 1 |
| P3 | 0 | 1 | x | x | x | x | x | x |
| P4 | x | x | 0 | 1 | x | x | x | x |
| P5 | x | x | x | x | 0 | 1 | x | x |
| P6 | x | x | x | x | x | x | 0 | 1 |

**Figure 6-26. L2 Cache Line Replacement Algorithm**

### 6.8.1 PLRU Bit Update Considerations

PLRU bit updates depend on which cache way was last accessed, as summarized in Table 6-24.

Table 6-24. PLRU Bit Update Algorithm

| Last Way Accessed | PLRU Bits | | | | | | |
|---|---|---|---|---|---|---|---|
| | P0 | P1 | P2 | P3 | P4 | P5 | P6 |
| 0 | 1 | 1 | — | 1 | — | — | — |
| 1 | 1 | 1 | — | 0 | — | — | — |
| 2 | 1 | 0 | — | — | 1 | — | — |
| 3 | 1 | 0 | — | — | 0 | — | — |
| 4 | 0 | — | 1 | — | — | 1 | — |
| 5 | 0 | — | 1 | — | — | 0 | — |
| 6 | 0 | — | 0 | — | — | — | 1 |
| 7 | 0 | — | 0 | — | — | — | 0 |

When an L2 line is invalidated, the PLRU bits are updated, marking the corresponding way as least-recently used. This causes the invalidated way to be selected as the next victim.

### 6.8.2 Allocation of Lines

The general PLRU algorithm described above must be modified to take into account special features of the L2 cache; namely SRAM regions, line locking, and stash-only regions. Each of these features reserves ways within each cache set such that some ways are not eligible for allocation/victimization by the general LRU algorithm.

To preserve the state of the ways that are set aside for other special functions, the PLRU pointers are modified by a mask that is a function of the L2 configuration registers, the lock bits in the cache status array, and initiator of the transaction. The mask effectively points the PLRU algorithm away from ways that are not to be considered for replacement.

L2 cache lines are locked through the status array lock bits. There are two lock bits for each way of each set (1024 sets by eight ways). These bits are set or cleared through special L2 controller commands.There are two sets of lock bits, one for instructions (I0–I7) and one for data (D0–D7) for every line. The lock bits act as a mask over the PLRU bits to determine victim selection. The PLRU bits are updated regardless of line locking.

Lock bits are used at allocate time to steer the PLRU algorithm away from selecting locked victims. In the following discussion, the eight lock bits for a particular set are called L0–L7.

```
Where Lock Way i: Li = Di | Ii, i=0…7 (Di = data lock, Ii = instruction lock)
```

An effective value of each PLRU bit is calculated as follows:

```
P0_eff = f(P0,L0,L1,L2,L3,L4,L5,L6,L7) = (L0 & L1 & L2 & L3) | (P0 & ~(L4 & L5 & L6 & L7))
P1_eff = f(P1,L0,L1,L2,L3) = (L0 & L1) | (P1 & ~(L2 & L3))
P2_eff = f(P2,L4,L5,L6,L7) = (L4 & L5) | (P2 & ~(L6 & L7))
```

```
P3_eff = f(P3,L0,L1) = L0 | (P3 & ~L1)
P4_eff = f(P4,L2,L3) = L2 | (P4 & ~L3)
P5_eff = f(P5,L4,L5) = L4 | (P5 & ~L5)
P6_eff = f(P6,L6,L7) = L6 | (P6 & ~L7)
```

These effective PLRU bits are used to select a victim, as indicated in Table 6-25.

**Table 6-25. PLRU-Based Victim Selection Mechanism**

| Way Selected | Effective PLRU State (Binary) | Reduced Logic Equation (using effective PLRU bits) |
|---|---|---|
| W0 | 00x0xxx | ~P0 & ~P1 & ~P3 |
| W1 | 00x1xxx | ~P0 & ~P1 & P3 |
| W2 | 01xx0xx | ~P0 & P1 & ~P4 |
| W3 | 01xx1xx | ~P0 & P1 & P4 |
| W4 | 1x0xx0x | P0 & ~P2 & ~P5 |
| W5 | 1x0xx1x | P0 & ~P2 & P5 |
| W6 | 1x1xxx0 | P0 & P2 & ~P6 |
| W7 | 1x1xxx1 | P0 & P2 & P6 |

## 6.9 L2 Cache Operation

This section describes the behavior of the L1 and L2 cache in response to various operations and in various configurations.

### 6.9.1 Initialization

#### 6.9.1.1 L2 Cache Initialization

After power-on reset the valid bits in the L2 cache status array are in random states. Therefore, it is necessary to perform a flash invalidate before using the array as an L2 cache. This is done by writing a one to the L2I field of the L2 control register (L2CTL). This can be done before or simultaneously with the write that enables the L2 cache. That is, the L2E and L2I bits of L2CTL can be set simultaneously. The L2I bit clears automatically, so no further writes are necessary.

#### 6.9.1.2 Memory-Mapped SRAM Initialization

After power-on reset the contents of the data and ECC arrays are random, so all SRAM data must be initialized before it is read. If the cache is initialized by the processor or any other device that uses sub-cache-line transactions, ECC error checking should be disabled during the initialization process to avoid false ECC errors generated during the read-modify-write process used for sub-cache-line writes to the SRAM array. This is done by setting the multi- and single-bit ECC error disable bits of the L2 error disable register (L2ERRDIS[MBECCDIS, SBECCDIS]). See Section 6.3.1.4.2, “Error Control and Capture Registers.” If the array is initialized by a DMA engine using cache-line writes, ECC checking can remain enabled during the initialization process.

### 6.9.2 Flash Invalidation of the L2 Cache

The L2 cache may be completely invalidated by setting the L2I bit of the L2 control register (L2CTL). Note that no data is lost in this process because the L2 cache is a write-through cache and contains no modified data. Flash invalidation of the cache is necessary when the cache is initially enabled and may be necessary to recover from some error conditions such as a tag parity error.

The invalidation process requires several cycles to complete. The L2I bit remains set during this procedure and is then cleared automatically when the procedure is complete. The L2 cache controller issues retries for all transactions on the e500 core complex bus while the flash invalidation process is in progress.

Note that the contents of memory-mapped SRAM regions of the data array are unaffected by a flash invalidation of the L2 cache regions of the array.

### 6.9.3 Managing Errors

#### 6.9.3.1 ECC Errors

An individual soft error that causes a single- or multi-bit ECC error can be cleared from the L2 array simply by performing a **dcbf** instruction on the address captured in the L2ERRADDR register. This invalidates the line in the L2 cache. When the load that caused the ECC error is performed again, the data is reallocated into the L2 with ECC bits set properly again.

If the threshold for single bit errors set in the L2ERRCTL register is exceeded, then the L2 cache should be flash invalidated to clear out all single-bit errors.

Note that no data is lost by **dcbf**s or flash invalidates, since the L2 cache is write-through and contains no modified data.

#### 6.9.3.2 Tag Parity Errors

A tag parity error must be fixed by flash invalidating the L2 cache. Note that a **dcbf** operation to the address that caused the error to be reported is not sufficient since a tag parity error is seen as an L2 miss and does not cause invalidation of the bad tag. Proper L2 operation cannot be guaranteed if an L2 tag parity error is not repaired by a flash invalidation of the entire array.

### 6.9.4 L2 Cache States

The L2 status array uses four bits for each line to determine the status of the line. Different combinations of these bits result in different L2 states. The status bits are as follows:

- Valid (V)
- Instruction locked (IL)
- Data locked (DL)
- Stale (T)

Table 6-26 shows L2 cache states. Note that these conventions are also used in Table 6-27.

Table 6-26. L2 Cache States

| V | T | IL | DL | L2 states |
|---|---|---|---|---|
| 0 | x | x | x | Invalid (I) |
| 1 | 0 | 0 | 0 | Exclusive (E) |
| 1 | 0 | 0 | 1 | Exclusive data locked (EDL) |
| 1 | 0 | 1 | 0 | Exclusive instruction locked (EIL) |
| 1 | 0 | 1 | 1 | Exclusive instruction and data locked (EL) |
| 1 | 1 | 0 | 0 | Stale (data invalid, locks invalid) (T) |
| 1 | 1 | 0 | 1 | Stale (data invalid, dlock valid) (TDL) |
| 1 | 1 | 1 | 0 | Stale (data invalid, ilock valid) (TIL) |
| 1 | 1 | 1 | 1 | Stale (data invalid, locks valid) (TL) |

## 6.9.5 L2 State Transitions

Table 6-27 lists state transitions for all e500 core-initiated transactions that change the L2 cache state. Core-initiated transactions caused when the core executes **msync**, **mbar**, **tlbivax**, or **tlbsync** do not change the L2 cache state. The table does not list initial L1 states for transactions that hit in the L1 (iL1 or dL1) and are not sent to the L2.

In the table, the heading ‘L2 hit’ indicates that the L2 provides (on a read) or captures (on a write) data for an existing line. Some entries list two final L1 states. L2 touch instructions never allocate into iL1 or dL1.

Note that if the L2 SRAM is disabled, the L2 initial and final states are always I and the L2 never hits. Similarly, if the L2 SRAM is in full memory-mapped SRAM mode, the L2 initial and final states are always I and the L2 never hits for addresses not in the memory-mapped SRAM address range. The L2 always hits for addresses in the enabled memory-mapped SRAM address ranges.

Table 6-27. State Transitions Due to Core-Initiated Transactions

| Source of Transaction | Initial States | | L2 Hit | Final States | | Comments |
|---|---|---|---|---|---|---|
| | L1 | L2 | | L1 | L2 | |
| Cacheable instruction fetch<br>icbtls_L1 | iL1<br>I | I/T | No | I/V | same | L2CTL[L2DO] = 1. L2 touch instructions not allocated in L1 |
| | | I | No | I/V | E | L2CTL[L2DO] = 0 |
| icbt_L2 | dL1<br>I,E | E/EL | Yes | I/V | same | |
| | | T | No | I/V | EL | L2CTL[L2DO] = 0. Restore locked line in L2 with valid data from bus |

**Table 6-27. State Transitions Due to Core-Initiated Transactions (continued)**

| Source of Transaction | Initial States | | L2 Hit | Final States | | Comments |
|---|---|---|---|---|---|---|
| | L1 | L2 | | L1 | L2 | |
| icbtls_L2 | dL1<br>I,E | I/T | No | I | same | L2CTL[L2DO] = 1 |
| | | E | Yes | I | I | L2CTL[L2DO] = 1 |
| | | EL | Yes | I | T | L2CTL[L2DO] = 1 |
| | | I | No | I | EL | L2CTL[L2DO] = 0 |
| | | E | Yes | I | EL | L2CTL[L2DO] = 0 |
| | | EL | Yes | I | same | L2CTL[L2DO] = 0 |
| | | T | No | I | EL | L2CTL[L2DO] = 0. Restore locked line in L2 with valid data from bus |
| Cache-inhibited instruction fetch | N/A | N/A | No | N/A | N/A | No L1/L2 effect |
| Cacheable load (4-state)<br>Cacheable **lwarx** (4-state)<br>dcbt_L1 (4-state)<br>dcbtls_L1 (4-state) | dL1<br>I | I/T | No | E | same | L2CTL[L2IO] = 1 |
| | | E | Yes | E | I | L2CTL[L2IO] = 1 |
| | | EL | Yes | E | T | L2CTL[L2IO] = 1 |
| | | I | No | E | E | L2CTL[L2IO] = 0 |
| | | E/EL | Yes | E | same | L2CTL[L2IO] = 0 |
| | | T | No | EL | EL | L2CTL[L2IO] = 0. Restore locked line in L2 with valid data from bus |
| Cache-inhibited load | N/A | N/A | No | N/A | N/A | No L1/L2 effect |
| Cache-inhibited **lwarx** | N/A | N/A | No | N/A | N/A | No L2 effect |
| Writeback Store | dL1<br>I | I/T | No | M | same | L2 allocates when a line is cast out of L1. |
| | | E | Yes | M | I | |
| | | EL | Yes | M | T | |
| Writeback **stwcx** | dL1<br>I | I/T | No | M | same | |
| | | E | Yes | M | I | |
| | | EL | Yes | M | T | |
| Cacheable load (3-state)<br>Cacheable **lwarx** (3-state)<br>dcbt_L1 (3-state)<br>dcbtls_L1 (3-state) | dL1<br>I | I | No | E/I | I | L2CTL[L2IO] = 1 |
| | | T | No | E/I | T | L2CTL[L2IO] = 1 |
| | | E | Yes | E/I | I | L2CTL[L2IO] = 1 |
| | | EL | Yes | E/I | T | L2CTL[L2IO] = 1 |
| | | I | No | E/I | E | L2CTL[L2IO] = 0 |
| dcbt_L2<br>dcbtst_L2 | dL1<br>I,E | E/EL | Yes | E/I | same | L2CTL[L2IO] = 0 |
| | | T | No | E/I | EL | L2CTL[L2IO] = 0. Restore locked line with valid data from bus |
| dcbtst_L1<br>dcbtstls_L1 | dL1<br>I | I/T | No | E | same | |
| | | E | Yes | E | I | |
| | | EL | Yes | E | T | |

### Table 6-27. State Transitions Due to Core-Initiated Transactions (continued)

| Source of Transaction | Initial States | | L2 Hit | Final States | | Comments |
|---|---|---|---|---|---|---|
| | L1 | L2 | | L1 | L2 | |
| dcbtls_L2<br>dcbtstls_L2 | dL1<br>I,E | I | No | I | I | L2CTL[L2IO] = 1 |
| | | T | No | I | T | L2CTL[L2IO] = 1 |
| | | E | Yes | I | I | L2CTL[L2IO] = 1 |
| | | EL | Yes | I | T | L2CTL[L2IO] = 1 |
| | | I | No | I | EL | L2CTL[L2IO] = 0 |
| | | E/EL | Yes | I | EL | L2CTL[L2IO] = 0 |
| | | T | No | I | EL | L2CTL[L2IO] = 0. Restore locked line with valid data from bus |
| Write-through store | dL1<br>I,E,M | I/T | No | same | I | |
| | | E/EL | Yes | same | same | Read-modify-write |
| Cache-inhibited store | N/A | I/E | No | N/A | I | Invalidate line |
| | | EL/T | No | N/A | T | Invalidate data, keep lock |
| Cache-inhibited **stwcx** | N/A | I/E | No | N/A | I | Invalidate line |
| | | EL/T | No | N/A | T | Invalidate data, keep lock |
| dcblc_L2<br>icblc_L2 | dL1<br>I,E,M | I/E | No | same | same | |
| | | EL | No | same | E | |
| | | T | No | same | I | |
| Victim castout<br>dcbt_L2<br>icbt_L2<br>dcbtst_L2<br>Snoop push | dL1<br>M | I/T | No | I | same | L2CTL[L2IO] = 1. If software sharing cache lines between instructions and data wishes to capture instruction lines in L2 with L2CTL[L2IO] = 1, it must perform **dcbst** to flush the line out of the dL1 before fetching it into L2. |
| | | I | No | I | E | L2CTL[L2IO] = 0 |
| | | E/EL | No | I | I/T | L2CTL[L2IO] = 1. |
| | | | Yes | I | Same | L2CTL[L2IO] = 0. |
| | | T | Yes | I | EL | L2CTL[L2IO = 0. |
| dcbtls_L2<br>icbtls_L2<br>dcbtstls_L2 | dL1<br>M | I | No | I | EL | An icbtls_L2 that hits modified in L1 cannot be distinguished from dcbtls_L2 and sets the L2 dlock bit. If software shares cache lines between instructions and data and wishes to set hillocks in L2, it must perform **dcbst** to flush the line out of the dL1 before locking it in L2. |
| | | E/EL/T | Yes | I | EL | |
| **dcbf**<br>**dcbst** | dL1<br>M | I/E/EL | No | I | I | |
| **dcbz**<br>**dcba** | dL1<br>I | I/E | No | M | I | |
| | | EL | No | M | T | |
| **dcbi** | dL1<br>I,E,M | I/<br>E/EL/T | No | I | I | |

**Table 6-27. State Transitions Due to Core-Initiated Transactions (continued)**

| Source of Transaction | Initial States | | L2 Hit | Final States | | Comments |
|---|---|---|---|---|---|---|
| | L1 | L2 | | L1 | L2 | |
| **dcbf**<br>**dcbst** | dL1 I,E | I/E/EL/T | No | I | I | |
| **icbi** | iL1 I,V | I/E/EL/T | No | I | I | |

Table 6-28 lists L2 cache state transitions for all system-initiated (non-core) transactions that change the L2. The transaction types and attributes listed follow MPX bus nomenclature, with the addition of write allocate (burst write with L2 cache allocation). Table 6-28 accounts for changes caused by L1 snoop pushes triggered by snoops, listed in Table 6-27.

**Table 6-28. State Transitions Due to System-Initiated Transactions**

| Transaction Type | Initial L2 State | Final L2 State | Comments |
|---|---|---|---|
| Read, snoop local processor | I/T | Same | — |
| | E | E | — |
| | EL | EL | — |
| Read, unlock L2 cache line | I/T | I | — |
| | E/EL | E | — |
| Write 32-byte | I/E/EL/T | I | Miss in cache external write (CEW) windows |
| | I | E | Hit in CEW window |
| | E/EL | Same | |
| | T | EL | |
| | I/E/EL/T | EL | Hit in CEW window (CEW lock attribute set) |
| Write < 32-byte | I/E | I | Miss in CEW windows |
| | EL/T | T | |
| | I/T | Same | Hit in CEW window (no data is written) (regardless of CEW lock attribute) |
| | E/EL | Same | Hit in CEW window |
| | E/EL | EL | Hit in CEW window (CEW lock attribute set) |
| Write, allocate L2 cache line 32-byte | I/E | E | Allocate regardless of CEW window |
| | EL/T | EL | |
| Write, allocate L2 cache line < 32-byte | I/T | Same | No data is written |
| | E/EL | Same | — |
| Write, allocate and lock L2 cache line 32-byte | I/E/EL/T | EL | Allocate and lock regardless of CEW window |
| Write, allocate and lock L2 cache line < 32-byte | I/T | Same | No data is written |
| | E/EL | EL | — |
| ATOMIC increment, decrement, set, and clear | I/E | I | Invalidate line |
| | EL/T | T | Invalidate data, keep lock |

### 6.9.6 Error Checking and Correcting (ECC)

The L2 cache supports error checking and correcting (ECC) for the data path between the core master and system memory. It detects all double-bit errors, detects all multi-bit errors within a nibble, and corrects all single-bit errors. Other errors may be detected, but are not guaranteed to be corrected or detected. Multiple-bit errors are always reported when error reporting is enabled. When a single-bit error occurs, the single-bit error counter register is incremented, and its value compared to the single-bit error trigger register. An error is reported when these values are equal. The single-bit error registers can be programmed such that minor memory faults are corrected and ignored, but double- or multi-bit errors generate an interrupt.

The syndrome encodings for the ECC code are shown in Table 6-29 and Table 6-30.

**Table 6-29. L2 Cache ECC Syndrome Encoding**

| Data Bit | Syndrome Bit | | | | | | | | Data Bit | Syndrome Bit | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0 | • | • | | | | | | • | 32 | | | • | • | | | | • |
| 1 | • | | • | | | | | • | 33 | | | • | | • | | | • |
| 2 | • | | | • | | | | • | 34 | • | | • | | • | | | |
| 3 | • | | | | • | | | • | 35 | | • | • | | • | | | |
| 4 | • | • | | | | • | | | 36 | | | • | • | | • | | |
| 5 | • | | • | | | • | | | 37 | | | • | | • | • | | |
| 6 | • | | | • | | • | | | 38 | • | | • | | • | • | | • |
| 7 | • | | | | • | • | | | 39 | | • | • | | • | • | | • |
| 8 | • | • | | | | | • | | 40 | | | • | • | | | • | |
| 9 | • | | • | | | | • | | 41 | | | • | | • | | • | |
| 10 | • | | | • | | | • | | 42 | • | | • | | • | | • | • |
| 11 | • | | | | • | | • | | 43 | | • | • | | • | | • | • |
| 12 | • | • | | | | • | • | • | 44 | | | • | • | | • | • | • |
| 13 | • | | • | | | • | • | • | 45 | | | • | | • | • | • | • |
| 14 | • | | | • | | • | • | • | 46 | • | | • | | • | • | • | |
| 15 | • | | | | • | • | • | • | 47 | | • | • | | • | • | • | |
| 16 | | • | • | | | | | • | 48 | | • | | | | • | • | |
| 17 | | • | | • | | | | • | 49 | | | • | | | • | • | |
| 18 | | • | | | • | | | • | 50 | | | | • | | • | • | |
| 19 | • | • | | | • | | | | 51 | • | | | | | • | • | |
| 20 | | • | • | | | • | | | 52 | | • | | | | • | | • |
| 21 | | • | | • | | • | | | 53 | | | • | | | • | | • |
| 22 | | • | | | • | • | | | 54 | | | | • | | • | | • |
| 23 | • | • | | | • | • | | • | 55 | • | | | | | • | | • |
| 24 | | • | • | | | | • | | 56 | | • | | | | | • | • |

**Table 6-29. L2 Cache ECC Syndrome Encoding (continued)**

| Data Bit | Syndrome Bit | | | | | | | | Data Bit | Syndrome Bit | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 25 | | • | | • | | | • | | 57 | | | • | | | | • | • |
| 26 | | • | | | • | | • | | 58 | | | | • | | | • | • |
| 27 | • | • | | | • | | • | • | 59 | • | | | | | | • | • |
| 28 | | • | • | | | • | • | • | 60 | | | | • | • | | • | |
| 29 | | • | | • | | • | • | • | 61 | • | | | • | • | | • | • |
| 30 | | • | | | • | • | • | • | 62 | | • | | • | • | | • | • |
| 31 | • | • | | | • | • | • | | 63 | | | • | • | • | | • | • |

**Table 6-30. L2 Cache ECC Syndrome Encoding (Check Bits)**

| Check Bit | Syndrome Bit | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0 | • | | | | | | | |
| 1 | | • | | | | | | |
| 2 | | | • | | | | | |
| 3 | | | | • | | | | |
| 4 | | | | | • | | | |
| 5 | | | | | | • | | |
| 6 | | | | | | | • | |
| 7 | | | | | | | | • |

# Part III
# Memory, Security, and I/O Interfaces

Part III defines the memory, security and I/O interfaces of the MPC8536E and it describes how these blocks interact with one another and with other blocks on the device. The following chapters are included:

- Chapter 7, "e500 Coherency Module," defines the e500v2 coherency module and how it facilitates communication between the e500v2 core complex, the L2 cache, and the other blocks that comprise the coherent memory domain of the MPC8536E.

  The ECM permits I/O-initiated transactions to snoop the core complex bus (CCB) of the e500v2 core to maintain coherency across cacheable local memory. It also provides a flexible, easily expandable switch-type structure for e500v2- and I/O-initiated transactions to be routed (dispatched) to target modules on the MPC8536E.
- Chapter 8, "DDR Memory Controller," describes the DDR2/DDR3 SDRAM memory controller of the MPC8536E. This fully programmable controller supports most DDR memories available today, including both buffered and unbuffered devices. The built-in error checking and correction (ECC) ensures very low bit-error rates for reliable high-frequency operation. Dynamic power management and auto-precharge modes simplify memory system design. Special features like ECC error injection support rapid system debug.
- Chapter 9, "Programmable Interrupt Controller (PIC)," describes the programmable interrupt controller (PIC) of the MPC8536E. The PIC is an OpenPIC-compliant interrupt controller that provides interrupt management and is responsible for receiving hardware-generated interrupts from different sources (both internal and external), prioritizing them and delivering them to the CPU for servicing.
- Chapter 10, "Security Engine (SEC) 3.0," describes the security controller of the MPC8536E. The SEC 3.0 off-loads computationally intensive security functions, such as key generation and exchange, authentication, and bulk encryption from the processor cores of the MPC8536E. It is optimized to process all cryptographic algorithms associated with IPsec, IKE, SSL/TLS, iSCSI, SRTP, 802.11i, 3G, A5/3 for GSM and EDGE, and GEA3 for GPRS.
- Chapter 11, "I$^2$C Interfaces," describes the inter-IC (IIC or I$^2$C) bus controllers of the MPC8536E. This synchronous, serial, bidirectional, multi-master bus allows two-wire connection of devices, such as microcontrollers, EEPROMs, real-time clock devices, A/D converters and LCDs. The MPC8536E powers up in boot sequencer mode which allows the I$^2$C1 controller to initialize configuration registers.
- Chapter 12, "DUART," describes the (dual) universal asynchronous receiver/transmitters (UARTs) which feature a PC16552D-compatible programming model. These independent UARTs are provided specifically to support system debugging.
- Chapter 13, "Enhanced Local Bus Controller," describes the enhanced local bus controller (eLBC). The main component of the enhanced local bus controller is its memory controller, which provides

a seamless interface to many types of memory devices and peripherals. The memory controller controls eight memory banks shared by a general-purpose chip-select machine (GPCM), a NAND flash control machine (FCM), and up to three user-programmable machines (UPMs). As such, it supports a minimal glue logic interface to SRAM, EPROM, Flash EPROM, burstable RAM, regular DRAM devices, extended data output DRAM devices, and other peripherals.

- Chapter 14, "Enhanced Three-Speed Ethernet Controllers," describes the two enhanced three-speed Ethernet controllers. These controllers support 10/100/1Gb Ethernet with a complete set of media-independent interface options including MII, RMII, GMII, RGMII, SGMII, TBI, and RTBI. Each controller provides very high throughput using a captive DMA channel and direct connection to the memory coherency module. The controllers provide two full-duplex FIFO interface modes and quality of service support and are backward compatible with PowerQUICC III TSEC controllers.
- Chapter 15, "DMA Controller," describes the four-channel general-purpose DMA controller of the MPC8536E. The DMA controller transfers blocks of data independent of the e500v2 core or external hosts. Data movement occurs among the local address space. The DMA controller has four high-speed channels. Both the e500 core and external masters can initiate a DMA transfer. All channels are capable of complex data movement and advanced transaction chaining.
- Chapter 16, "PCI Bus Interface," describes the PCI interface, which complies with the *PCI Local Bus Specification*, Rev. 2.3. This chapter provides a basic description of PCI bus operations. The specific emphasis is directed at how this device implements the PCI specification.
- Chapter 17, "PCI Express Interface Controller," describes the three PCI Express controllers of the MPC8536E. Each controller is compliant with the *PCI Express Base Specification Revision 1.0a*. The physical layer of these controllers operate at 2.5 Gbaud per lane. Configuration options allow multiple width configurations among the three controllers.
- Chapter 18, "Enhanced Serial Peripheral Interface," describes the serial peripheral interface (SPI), which allows data exchange between MPC85xx family devices. It can also be used to communicate with peripheral devices such as EEPROMs, real-time clocks, A/D converters, and ISDN devices.
- Chapter 19, "SATA Controller," describes the serial ATA controllers of the MPC8536E.
- Chapter 20, "Enhanced Secure Digital Host Controller," describes the enhanced SD Host Controller, which provides an interface between the host system and SD and MMC cards. It provides a functional description of the major system blocks and includes command information for the host.
- Chapter 21, "Universal Serial Bus Interfaces," describes the three universal serial bus (USB) interfaces. The USB module is a USB 2.0-compliant serial interface engine for implementing a USB interface. The registers and data structures are based on the *Enhanced Host Controller Interface Specification for Universal Serial Bus* (EHCI) from Intel Corporation. The USB dual-role modules can act as a host or as a device on the USB bus.
- Chapter 22, "General Purpose I/O (GPIO)," describes the general-purpose input and output signals of the MPC8536E.

# Chapter 7
# e500 Coherency Module

## 7.1 Introduction

The e500 coherency module (ECM) provides a flexible switching structure for routing e500- and I/O-initiated transactions to target modules on the device. Figure 7-1 shows a high-level block diagram of the ECM.

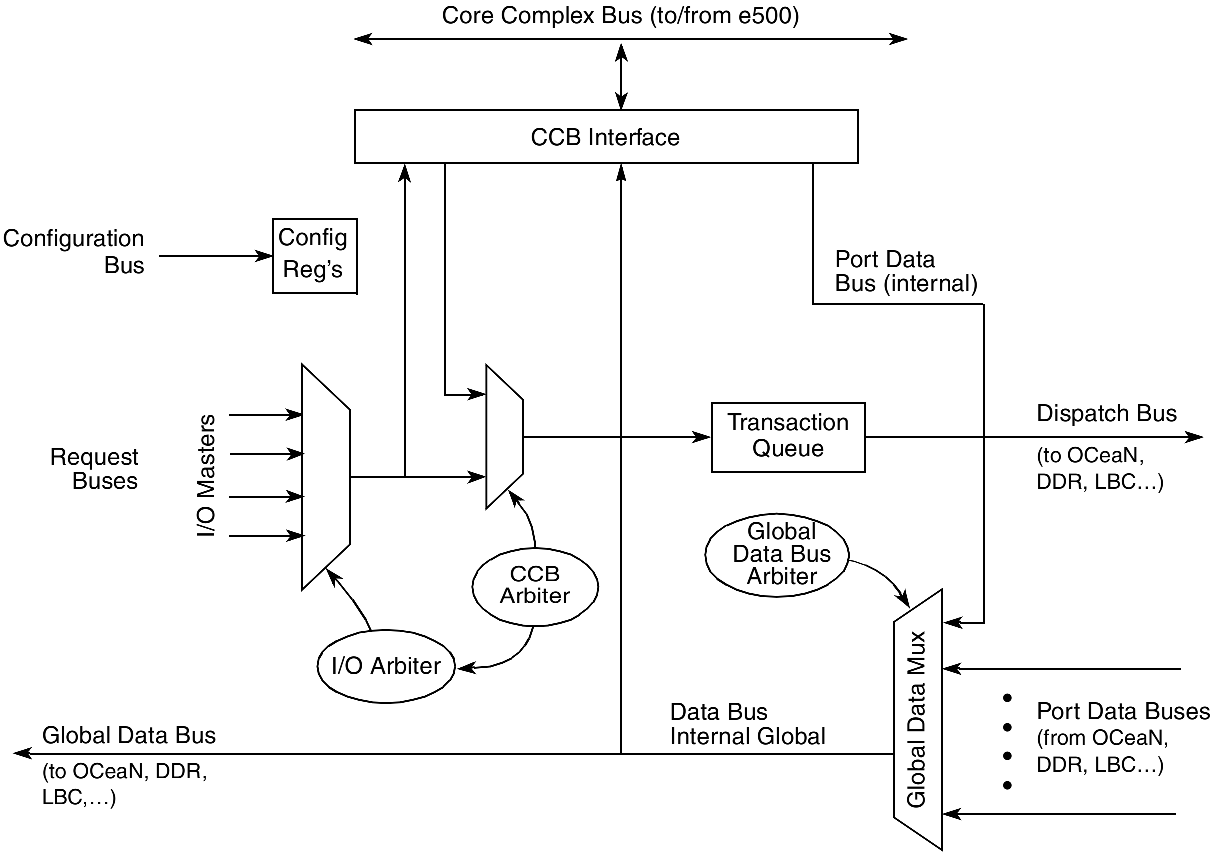


**Figure 7-1. e500 Coherency Module Block Diagram**

### 7.1.1 Overview

The ECM routes transactions initiated by the e500 core to the appropriate target interface on the device. In a manner analogous to a bridging router in a local area network, the ECM forwards I/O-initiated transactions that are tagged with the global attribute onto the core complex bus (CCB). This allows on-chip caches to snoop these transactions as if they were locally initiated and to take actions to maintain coherency across cacheable memory.

### 7.1.2 Features

The ECM includes these distinctive features:

- Support for the e500 core and an L2/SRAM on the CCB, including a CCB arbiter.
- It sources a 64-bit data bus for returning read data from the ECM to the e500 core and routing write data from the ECM to the L2/SRAM. It sinks a 128-bit data bus for receiving data from the L2/SRAM and a 128-bit write data bus from the e500 core.
- Four connection points for I/O initiating (mastering into the device) interfaces. The ECM supports five connection points for I/O targets. The DDR memory controller, enhanced local bus, OCeaN targets, and configuration register access block all have a target port connection to the ECM.
- Split transaction support—separate address and data tenures allow for pipelining of transactions and out-of-order data tenures between initiators and targets.
- Proper ordering of I/O-initiated transactions.
- Speculative read bus for low-latency dispatch of reads to the DDR controller.
- Low-latency path for returning read data from DDR to the e500 core.
- Error registers trap transactions with invalid addresses. Errors can be programmed to generate interrupts to the e500 core, as described in the following sections:
  - Section 7.2.1.5, “ECM Error Detect Register (EEDR)”
  - Section 7.2.1.6, “ECM Error Enable Register (EEER)”
  - Section 7.2.1.7, “ECM Error Attributes Capture Register (EEATR)”
  - Section 7.2.1.8, “ECM Error Low Address Capture Register (EELADR)”
  - Section 7.2.1.9, “ECM Error High Address Capture Register (EEHADR)”
- Errors from reading I/O devices terminate with data sent to the master with a corrupt attribute. If the master is the e500 core, the ECM asserts *core_fault_in* to the core, which causes the core to generate a machine check interrupt, unless it is disabled (by clearing HID1[RFXE]). If RFXE is zero and one of these errors occurs, appropriate interrupts must be enabled to ensure that an interrupt is generated. See Section 5.2, “e500 Core Integration and the Core Complex Bus (CCB),” and the *PowerPC™ e500 Core Family Reference Manual*.

## 7.2 Memory Map/Register Definition

Table 7-1 shows the ECM's memory map. Undefined 4-byte address spaces within offset 0x000–0xFFF are reserved.

In this table and in the register figures and field descriptions, the following access definitions apply:

- Reserved fields are always ignored for the purposes of determining access type.
- R/W, R, and W (read/write, read only, and write only) indicate that all the non-reserved fields in a register have the same access type.
- w1c indicates that all of the non-reserved fields in a register are cleared by writing ones to them.
- Mixed indicates a combination of access types.
- Special is used when no other category applies. In this case the register figure and field description table should be read carefully.

**Table 7-1. ECM Memory Map**

| Local Memory Offset | Register | Access | Reset | Section/page |
|---|---|---|---|---|
| 0x0_1000 | EEBACR—ECM CCB address configuration register | R/W | 0x0000_0003 | 7.2.1.1/7-3 |
| 0x0_1010 | EEBPCR—ECM CCB port configuration register | R/W | 0x0*n*00_0000 | 7.2.1.2/7-4 |
| 0x0_1BF8 | ECM IP Block Revision Register 1 | R | 0x0001_0000 | 7.2.1.3/7-5 |
| 0x0_1BFC | ECM IP Block Revision Register 2 | R | 0x0000_0000 | 7.2.1.4/7-5 |
| 0x0_1E00 | EEDR—ECM error detect register | w1c | 0x0000_0000 | 7.2.1.5/7-6 |
| 0x0_1E08 | EEER—ECM error enable register | R/W | 0x0000_0000 | 7.2.1.6/7-7 |
| 0x0_1E0C | EEATR—ECM error attributes capture register | R | 0x0000_0000 | 7.2.1.7/7-7 |
| 0x0_1E10 | EELADR—ECM error low address capture register | R | 0x0000_0000 | 7.2.1.8/7-8 |
| 0x0_1E14 | EEHADR—ECM error high address capture register | R | 0x0000_0000 | 7.2.1.9/7-9 |

### 7.2.1 Register Descriptions

This section consists of detailed descriptions of those registers summarized in Table 7-1. Note that these registers are shown in big-endian format.

### 7.2.1.1 ECM CCB Address Configuration Register (EEBACR)

The ECM CCB address configuration register, shown in Figure 7-2, controls arbitration and streaming policies for the CCB.

Offset 0x0_1000 Access: Read/Write

| | 0 … 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|
| R<br>W | — | A_STRM_DIS | CORE_STRM_DIS | A_STRM_CNT | |
| Reset | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 | 0 | 1 | 1 |

**Figure 7-2. ECM CCB Address Configuration Register (EEBACR)**

Table 7-2 describes the EEBACR fields.

**Table 7-2. EEBACR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–27 | — | Reserved |
| 28 | A_STRM_DIS | Controls whether the ECM allows any streaming to occur.<br>0 Streaming is enabled.<br>1 Streaming is disabled. |
| 29 | CORE_STRM_DIS | With A_STRM_DIS, controls whether the e500 core can stream commands onto the CCB. A_STRM_DIS and CORE_STRM_DIS must both be cleared for the e500 core to be enabled to stream address tenures that it masters.<br>0 Stream address tenures initiated by the e500 core, provided A_STRM_DIS is cleared.<br>1 Streaming of address tenures initiated by the e500 core not allowed. |
| 30–31 | A_STRM_CNT | Stream count. Specifies the maximum number of transactions that any master can stream (issue sequentially without preemption) on the CCB following an initial transaction.<br>00 Reserved<br>01 One transaction can be streamed with the initial transaction.<br>10 Two transactions can be streamed with the initial transaction.<br>11 Three transactions can be streamed with the initial transaction. Default. |

### 7.2.1.2 ECM CCB Port Configuration Register (EEBPCR)

The ECM CCB port configuration register (EEBPCR) is shown in Figure 7-3.

Offset 0x0_1010 Access: Read/Write

| | 0–6 | 7 | 8–28 | 29 | 30–31 |
|---|---|---|---|---|---|
| R<br>W | — | CPU_EN | — | CPU_RD_HI_DIS | CPU_PRI |
| Reset | 0 0 0 0 0 0 0 | *n* | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 | 0 0 |

**Figure 7-3. ECM CCB Port Configuration Register (EEBPCR)**

Table 7-3 describes EEBPCR fields.

**Table 7-3. EEBPCR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–6 | — | Reserved |
| 7 | CPU_EN | CPU port enable. Controls boot holdoff mode when the device is an agent of an external host. Specifies whether the e500 core (CPU) port is enabled to run transactions on the CCB. The CPU boot configuration power-on reset pin (cfg_cpu_boot) determines the initial value of this bit. If the pin is sampled as a logic 1 at the negation of reset, the CPU is enabled to boot at the end of the POR sequence. Otherwise, the CPU cannot fetch its boot vector until an external host sets the CPU_EN bit.<br>0 Boot holdoff mode. CPU arbitration is disabled on the CCB and no bus grants are issued.<br>1 CPU is enabled and receives bus grants in response to bus requests for the boot vector.<br>After this bit is set, it should not be cleared by software. It is not intended to dynamically enable and disable CPU operation. It is only intended to end boot holdoff mode. See Section 4.4.3.10, “CPU Boot Configuration,” for more information. |
| 8–28 | — | Reserved |

**Table 7-3. EEBPCR Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 29 | CPU_RD_HI_DIS | Identifies which read queue of DDR targets is assigned to the e500 core (CPU) port's read transactions (in understressed system).<br>0 Read high queue (higher bandwidth DDR queue) is assigned for the e500 core's read transactions<br>1 Read low queue (lower bandwidth DDR queue) is assigned for the e500 core's read transactions |
| 30–31 | CPU_PRI | Specifies the priority level of the e500 core 0 (CPU) port. This priority level is used to determine whether a particular port's bus request can cause the CCB arbiter to terminate another port's streaming of address tenures.<br>00 Lowest priority level<br>01 Second lowest priority level<br>10 Highest priority level<br>11 Reserved |

### 7.2.1.3 ECM IP Block Revision Register 1 (EIPBRR1)

The ECM IP block revision register 1 is shown in Figure 7-4.

Offset 0x0_1BF8 Access: Read only

| | 0–15 | 16–23 | 24–31 |
|---|---|---|---|
| R | IP_ID | IP_MJ | IP_MN |
| W | | | |
| Reset | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 |

**Figure 7-4. ECM IP Block Revision Register 1 (EIPBRR1)**

Table 7-4 describes EIPBRR1 fields.

**Table 7-4. EIPBRR1 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–15 | IP_ID | IP block ID |
| 16–23 | IP_MJ | Major revision |
| 24–31 | IP_MN | Minor revision |

### 7.2.1.4 ECM IP Block Revision Register 2 (EIPBRR2)

The ECM IP block revision register 2 is shown in Figure 7-5.

Offset 0x0_1BFC Access: Read only

| | 0–7 | 8–15 | 16–23 | 24–31 |
|---|---|---|---|---|
| R | — | IP_INT | — | IP_CFG |
| W | | | | |
| Reset | All zeros | | | |

**Figure 7-5. ECM IP Block Revision Register 2 (EIPBRR2)**

Table 7-5 describes EIPBRR2 fields.

**Table 7-5. EIPBRR2 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–7 | — | Reserved |
| 8–15 | IP_INT | IP block integration options |
| 16–23 | — | Reserved |
| 24–31 | IP_CFG | IP block configuration options |

### 7.2.1.5 ECM Error Detect Register (EEDR)

The ECM error detect register (EEDR) is shown in Figure 7-6.

Offset 0x0_1E00 Access: w1c

| | 0 | 1–30 | 31 |
|---|---|---|---|
| R | MULT_ERR | — | LAE |
| W | w1c | | w1c |
| Reset | All zeros | | |

**Figure 7-6. ECM Error Detect Register (EEDR)**

Table 7-6 describes EEDR fields.

**Table 7-6. EEDR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | MULT_ERR | Multiple error. Indicates the occurrence of multiple errors of the same type. Write 1 to clear.<br>0 Multiple errors of the same type were not detected.<br>1 Multiple errors of the same type were detected. |
| 1–30 | — | Reserved |
| 31 | LAE | Local access error. Write 1 to clear. Two cases can generate LAEs:<br>• Transaction does not map to any target. In this case the ECM injects read responses (with the corrupt attribute set) and write data is dropped. Note that a read that attempts to access an unmapped target causes the assertion of *core_fault_in*, which causes the core to generate a machine check interrupt, unless it is disabled (by clearing HID1[RFXE]). If RFXE is zero and this error occurs, EEER[LAEE] must be set to ensure that an interrupt is generated. For more information, see Section 5.2, “e500 Core Integration and the Core Complex Bus (CCB),” and the *PowerPC™ e500 Core Family Reference Manual*.<br>• Source and target IDs indicate that an OCN port initiated a transaction that targets an OCN port. This loopback behavior can result from programming errors where inbound ATMU window targets are inconsistent with targets configured in the local access windows for a given address range. For this type of LAE, the dispatch (to OCN target in this case) is not screened off; the LAE error is reported, but the transaction is still sent to its OCN target.<br>0 Local access error has not occurred.<br>1 Local access error occurred. |

### 7.2.1.6 ECM Error Enable Register (EEER)

The ECM error enable register (EEER) shown in Figure 7-7 enables the reporting of error conditions to the e500 core through the internal $\overline{int}$ interrupt signal.

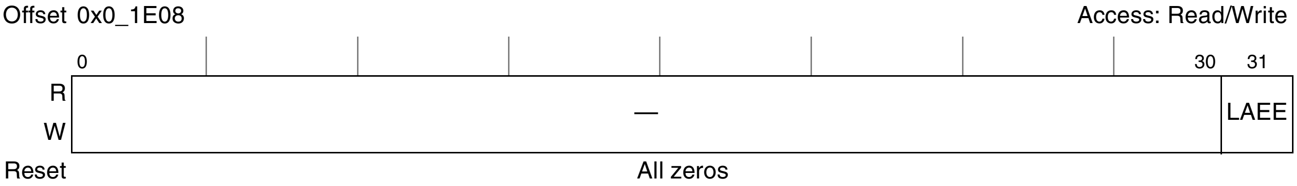


**Figure 7-7. ECM Error Enable Register (EEER)**

Table 7-7 describes EEER fields.

**Table 7-7. EEER Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–30 | — | Reserved |
| 31 | LAEE | Local access error enable. Note that a read that attempts to access an unmapped target causes the assertion of *core_fault_in*, which causes the core to generate a machine check interrupt, unless it is disabled (by clearing HID1[RFXE]). If HID1[RFXE] is zero and this error occurs, LAEE must be set to ensure that an interrupt is generated. For more information, see Section 5.2, "e500 Core Integration and the Core Complex Bus (CCB)," and the *PowerPC™ e500 Core Family Reference Manual*."<br>0 Disable reporting local access errors as interrupts.<br>1 Enable reporting local access errors as interrupts. |

### 7.2.1.7 ECM Error Attributes Capture Register (EEATR)

The ECM error attributes capture register (EEATR) is shown in Figure 7-8.

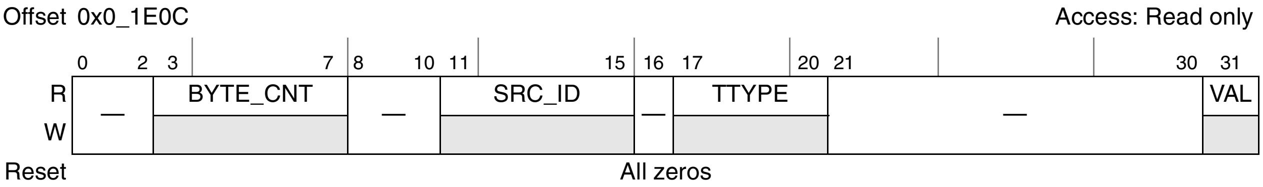


**Figure 7-8. ECM Error Attributes Capture Register (EEATR)**

Table 7-8 describes EEATR fields.

**Table 7-8. EEATR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–2 | — | Reserved |
| 3–7 | BYTE_CNT | Byte count. Specifies the transaction byte count.<br>00000 32 bytes<br>00001 1 byte<br>00010 2 bytes<br>00100 4 bytes<br>01000 8 bytes<br>10000 16 bytes |
| 8–10 | — | Reserved |

**Table 7-8. EEATR Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 11–15 | SRC_ID | Source ID. Specifies the source device mastering the transaction.<br>00000 PCI interface<br>00001 PCI Express 2<br>00010 PCI Express 1<br>00011 PCI Express 3<br>00100 Enhanced local bus<br>00101 USB1<br>00110 Reserved<br>00111 Security<br>01000 SATA2<br>01001 USB3<br>01010 Boot sequencer<br>01011 eSDHC<br>01100 Reserved<br>01101 SATA1<br>01110 Reserved<br>01111 DDR controller<br>10000 Processor (instruction)<br>10001 Processor (data)<br>10010–10011 Reserved<br>10100 USB2<br>10101 DMA<br>10110 Reserved<br>10111 SAP<br>11000 eTSEC1<br>11001 Reserved<br>11010 eTSEC3<br>11010–11111 Reserved |
| 16 | — | Reserved |
| 17–20 | TTYPE | Transaction type. Defined as follows:<br>0000 Write<br>0001 Reserved<br>0010 Write with allocate<br>0011 Write with allocate with lock<br>0100 Address only transaction<br>0101–0111Reserved<br>1000 Read<br>1001 Read with unlock<br>101x Reserved<br>1100 Read with clear atomic<br>1101 Read with set atomic<br>1110 Read with decrement atomic<br>1111 Read with increment atomic |
| 21–30 | — | Reserved |
| 31 | VAL | Register data valid.<br>0 ECM error attribute capture register does not contain valid information.<br>1 ECM error attribute capture register contains valid information. |

### 7.2.1.8 ECM Error Low Address Capture Register (EELADR)

The ECM error low address capture register (EELADR) is shown in Figure 7-9.

Offset 0x0_1E10 Access: Read only

| | 0 … 31 |
|---|---|
| R | ADDR |
| W | |
| Reset | All zeros |

**Figure 7-9. ECM Error Low Address Capture Register (EELADR)**

Table 7-9 describes EELADR fields.

**Table 7-9. EELADR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–31 | ADDR | Address. Specifies the lower-order 32 bits of the 36-bit address of the transaction. Qualified by EEATR[VAL]. |

### 7.2.1.9 ECM Error High Address Capture Register (EEHADR)

The ECM error high address capture register (EEHADR) is shown in Figure 7-10.

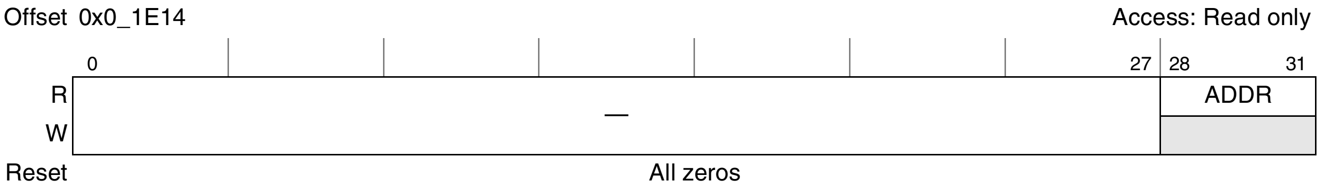


**Figure 7-10. ECM Error High Address Capture Register (EEHADR)**

Table 7-10 describes EEHADR fields.

**Table 7-10. EEHADR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–27 | — | Reserved |
| 28–31 | ADDR | Address. Specifies the high-order 4 bits of the 36-bit address of the transaction. Qualified by EEATR[VAL]. |

## 7.3 Functional Description

The following is a very general discussion of ECM operation.

### 7.3.1 I/O Arbiter

Figure 7-1 shows the I/O arbiter block that manages I/O-initiated address tenure requests arriving on the request buses. Four request buses compete for access to the ECM, which can only process one request at a time. The ECM uses two factors to select the winning request bus: the primary factor is requested bandwidth and the secondary factor is longest waiting/least recently granted status. By default all requesters start requesting low levels of bandwidth. A starvation avoidance algorithm ensures that low bandwidth requesters make forward progress in the presence of high bandwidth requesters.The transaction from the winning request bus competes with e500 core requests for the CCB and entry into the transaction queue.

### 7.3.2 CCB Arbiter

Figure 7-1 shows the CCB arbiter block coordinating the entry of new transactions into the ECM's transaction queue. It handles arbitration for requests to use the CCB from the e500 core and the winning request bus and consequently controls when these new transactions can enter the transaction queue.

Because the CCB bus operates most efficiently when it streams commands from one initiator, the CCB arbiter alternates grants between streams of transactions from the e500 core and from the winner of the I/O arbiter. The length of a stream (number of back-to-back transactions) is limited by the A_STRM_CNT field in the EEBACR register. However, the arbiter also uses the priority of the requests to limit streaming. If the priority of a new request is higher than that of a stream in progress, then the higher priority transaction interrupts the other stream. The priority of e500 transactions is set by the CPU_PRI field in

EEBPCR register. Depending how the CPU_RD_HI_DIS field in EEBPCR register is set, read transactions from the e500 core are initially assigned to either the higher- or lower-bandwidth queue of the DDR target.

### 7.3.3 Transaction Queue

The ECM's transaction queue performs four basic functions: arbitration across the e500 core and I/O masters, target mapping and dispatching, enforcement of ordering, and enforcement of coherency. The address of each transaction is compared against each local access window, and the transaction is then routed to the appropriate target interface associated with the local access window that the address hits within. Even though the CCB and ECM allow the pipelining of transactions, the address tenures of all transactions issued from I/O masters (masters other than the e500 core) may still be ordered. For those transactions accessing address space marked as snoopable, or space that may be cached by the e500 core, the ECM enforces coherency, snooping those transactions on the CCB, and taking castouts from the e500 core as is necessary.

### 7.3.4 Global Data Multiplexor

Figure 7-1 shows how the global data multiplexor takes data bus connections and multiplexes them onto one 128-bit global data bus. The global data mux allows initiators of write transactions to route data to their targets and read targets to return data to the initiators.

### 7.3.5 CCB Interface

Figure 7-1 shows the CCB interface for both CCB address and data tenures. This interface formats CCB address tenures for the ECM transaction queue. It also contains the queueing and buffering needed to manage outstanding CCB data tenures. The buffers receive e500 core-initiated write and I/O-initiated read data (that hit in the L2/SRAM module) from the e500 write (128-bit wide) and read (128-bit wide) data buses and route them through the global data mux to the global data bus. The buffers also receive e500 core-initiated read and I/O-initiated write data (that hit in the L2/SRAM module) from the global data bus and forward them onto the CCB data bus (64 bits).

## 7.4 Initialization/Application Information

If the e500 core is used to initialize the device, the CPU boot configuration power-on reset pin should be pulled high to initially set EEBPCR[CPU_EN]. See Chapter 4, "Reset, Clocking, and Initialization," for more information on power-up reset initialization.

If any device other than the e500 core (such as PCI Express) is used to initialize the device, the CPU boot configuration power-on reset pin should be pulled low to initially clear EEBPCR[CPU_EN]. This prevents the e500 core from accessing any configuration registers or local memory space during initialization. However, in any such system, one step near the end of the initialization routine must set EEBPCR[CPU_EN] to re-enable the e500 core. Note that for basic functionality, EEBPCR[CPU_EN] is the only field that must be written (provided a device other than the e500 core is used to initialize the device) in the ECM.

EEBPCR[CPU_PRI] specifies the priority level associated with all e500 core initiated transactions. This value allows users running time-critical applications to adjust the average response latency of transactions initiated by the core compared to those initiated by I/O masters. This priority level affects whether e500 core requests can interrupt the streaming of address tenures initiated by (the ECM on behalf of) I/O masters. Only transactions with a priority greater than the current CCB transaction can interrupt streaming. The higher the core's priority, the lower the average latency needed for it to obtain bus grants from the ECM, because it can interrupt lower priority streaming. The default value of zero gives all core-initiated transactions the lowest priority, which prevents the core from interrupting I/O master transaction streams.

EEBACR[A_STRM_CNT] allows users to balance response latency with throughput and should prove useful in tuning systems with multiple time-critical tasks. The default value of 0b11 causes the ECM to attempt to stream as many as four transactions initiated from the same CCB master. Increasing this value increases the maximum number of transactions that may be streamed together from any one CCB master. Raising this value can increase throughput for high priority transactions, but may increase latency for lower priority transactions from another CCB master. Note that the e500 core must also have streaming enabled (through HID1[ASTME]) for the CCB to stream.

# Chapter 8
DDR Memory Controller

## 8.1 Introduction

The fully programmable DDR SDRAM controller supports most JEDEC standard x8, x16, or x32 DDR2 and DDR3 memories available. In addition, unbuffered and registered DIMMs are supported. However, mixing different memory types or unbuffered and registered DIMMs in the same system is not supported. Built-in error checking and correction (ECC) ensures very low bit-error rates for reliable high-frequency operation. Dynamic power management and auto-precharge modes simplify memory system design. A large set of special features, including ECC error injection, support rapid system debug.

**NOTE**

In this chapter, the word ‘bank’ refers to a physical bank specified by a chip select; ‘logical bank’ refers to one of the four or eight sub-banks in each SDRAM chip. A sub-bank is specified by the 2 or 3 bits on the bank address (MBA) pins during a memory access.

Figure 8-1 is a high-level block diagram of the DDR memory controller with its associated interfaces. Section 8.5, "Functional Description," contains detailed figures of the controller.

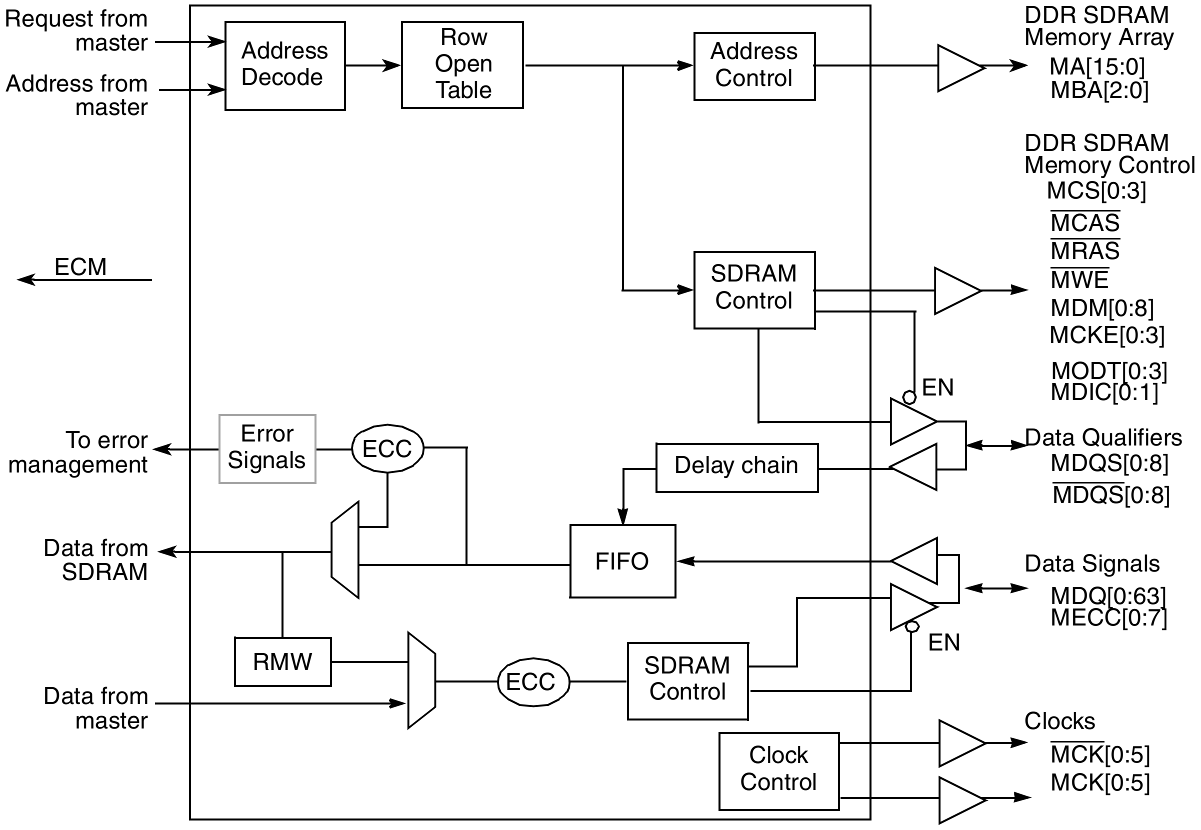


**Figure 8-1. DDR Memory Controller Simplified Block Diagram**

## 8.2 Features

The DDR memory controller includes these distinctive features:

- Support for DDR2 and DDR3 SDRAM
- 64-/72-bit SDRAM data bus. 32-/40-bit SDRAM for DDR2 and DDR3
- Programmable settings for meeting all SDRAM timing parameters
- The following SDRAM configurations are supported:
  - As many as four physical banks (chip selects), each bank independently addressable
    - 64-Mbit to 4-Gbit devices depending on internal device configuration with x8/x16/x32 data ports (no direct x4 support)
  - Unbuffered and registered DIMMs
- Chip select interleaving support
- Partial array self refresh support
- Support for data mask signals and read-modify-write for sub-double-word writes. Note that a read-modify-write sequence is only necessary when ECC is enabled.

- Support for double-bit error detection and single-bit error correction ECC (8-bit check word across 64-bit data)
- Support for address parity for registered DIMMs
- Open page management (dedicated entry for each logical bank)
- Automatic DRAM initialization sequence or software-controlled initialization sequence
- Automatic DRAM data initialization
- Write leveling supported for DDR3 memories
- Support for up to eight posted refreshes
- Memory controller clock frequency of two or four times the SDRAM clock with support for sleep power management
- Support for error injection

### 8.2.1 Modes of Operation

The DDR memory controller supports the following modes:

- Dynamic power management mode. The DDR memory controller can reduce power consumption by negating the SDRAM CKE signal when no transactions are pending to the SDRAM.
- Auto-precharge mode. Clearing DDR_SDRAM_INTERVAL[BSTOPRE] causes the memory controller to issue an auto-precharge command with every read or write transaction. Auto-precharge mode can be enabled for separate chip selects by setting CS*n*_CONFIG[AP_*n*_EN].

## 8.3 External Signal Descriptions

This section provides descriptions of the DDR memory controller’s external signals. It describes each signal’s behavior when the signal is asserted or negated and when the signal is an input or an output.

### 8.3.1 Signals Overview

Memory controller signals are grouped as follows:

- Memory interface signals
- Clock signals
- Debug signals

Table 8-1 shows how DDR memory controller external signals are grouped. The device hardware specification has a pinout diagram showing pin numbers. It also lists all electrical and mechanical specifications.

**Table 8-1. DDR Memory Interface Signal Summary**

| Name | Function/Description | Reset | Pins | I/O |
|---|---|---|---|---|
| $\overline{\text{MAPAR\_ERR}}$ | Address parity error | One | 1 | I |
| MAPAR_OUT | Address parity out | Zero | 1 | O |
| MDQ[0:63] | Data bus | All zeros | 64 | I/O |
| MDQS[0:8] | Data strobes | All zeros | 9 | I/O |
| $\overline{\text{MDQS}}$[0:8] | Complement data strobes | All ones | 9 | I/O |
| MECC[0:7] | Error checking and correcting | All zeros | 8 | I/O |
| $\overline{\text{MCAS}}$ | Column address strobe | One | 1 | O |
| MA[15:0] | Address bus | All zeros | 16 | O |
| MBA[2:0] | Logical bank address | All zeros | 3 | O |
| $\overline{\text{MCS}}$[0:3] | Chip selects | All ones | 4 | O |
| $\overline{\text{MWE}}$ | Write enable | One | 1 | O |
| $\overline{\text{MRAS}}$ | Row address strobe | One | 1 | O |
| MDM[0:8] | Data mask | All zeros | 9 | O |
| MCK[0:5] | DRAM clock outputs | All zeros | 6 | O |
| $\overline{\text{MCK}}$[0:5] | DRAM clock outputs (complement) | All zeros | 6 | O |
| MCKE[0:3] | DRAM clock enable | All zeros | 4 | O |
| MODT[0:3] | DRAM on-die termination external control. | All zeros | 4 | O |
| MDVAL | Memory debug data valid | Zero | 1 | O |
| MSRCID[0:4] | Memory debug source ID | All zeros | 5 | O |
| MDIC[0:1] | Driver impedance calibration | b10 | 2 | I/O |

Table 8-2 shows the memory address signal mappings.

Table 8-2. Memory Address Signal Mappings

| Signal Name (Outputs) | | JEDEC DDR DIMM Signals (Inputs) |
|---|---|---|
| msb | MA15 | A15 |
| | MA14 | A14 |
| | MA13 | A13 |
| | MA12 | A12 |
| | MA11 | A11 |
| | MA10 | A10 (AP for DDR)[1] |
| | MA9 | A9 |
| | MA8 | A8 (alternate AP for DDR)[2] |
| | MA7 | A7 |
| | MA6 | A6 |
| | MA5 | A5 |
| | MA4 | A4 |
| | MA3 | A3 |
| | MA2 | A2 |
| | MA1 | A1 |
| lsb | MA0 | A0 |
| msb | MBA2 | MBA2 |
| | MBA1 | MBA1 |
| lsb | MBA0 | MBA0 |

[1] Auto-precharge for DDR signaled on A10 when DDR_SDRAM_CFG[PCHB8] = 0

[2] Auto-precharge for DDR signaled on A8 when DDR_SDRAM_CFG[PCHB8] = 1

## 8.3.2 Detailed Signal Descriptions

The following sections describe the DDR SDRAM controller input and output signals, the meaning of their different states, and relative timing information for assertion and negation.

### 8.3.2.1 Memory Interface Signals

Table 8-3 describes the DDR controller memory interface signals.

**Table 8-3. Memory Interface Signals—Detailed Signal Descriptions**

| Signal | I/O | Description | |
|---|---|---|---|
| MDQ[0:63] | I/O | Data bus. Both input and output signals on the DDR memory controller. | |
| | O | As outputs for the bidirectional data bus, these signals operate as described below. | |
| | | **State Meaning** | Asserted/Negated—Represent the value of data being driven by the DDR memory controller. |
| | | **Timing** | Assertion/Negation—Driven coincident with corresponding data strobes (MDQS) signal.<br>High impedance—No READ or WRITE command is in progress; data is not being driven by the memory controller or the DRAM. |
| | I | As inputs for the bidirectional data bus, these signals operate as described below. | |
| | | **State Meaning** | Asserted/Negated—Represents the state of data being driven by the external DDR SDRAMs. |
| | | **Timing** | Assertion/Negation—The DDR SDRAM drives data during a READ transaction.<br>High impedance—No READ or WRITE command in progress; data is not being driven by the memory controller or the DRAM. |
| MDQS[0:8]/ $\overline{\text{MDQS}}$[0:8] | I/O | Data strobes. Inputs with read data, outputs with write data. | |
| | O | As outputs, the data strobes are driven by the DDR memory controller during a write transaction. The memory controller always drives these signals low unless a read has been issued and incoming data strobes are expected. This keeps the data strobes from floating high when there are no transactions on the DRAM interface. | |
| | | **State Meaning** | Asserted/Negated—Driven high when positive capture data is transmitted and driven low when negative capture data is transmitted. Centered in the data “eye” for writes; coincident with the data eye for reads. Treated as a clock. Data is valid when signals toggle. See Table 8-50 for byte lane assignments. |
| | | **Timing** | Assertion/Negation—If a WRITE command is registered at clock edge *n*, data strobes at the DRAM assert centered in the data eye on clock edge *n* + 1. See the JEDEC DDR SDRAM specification for more information. |
| | I | As inputs, the data strobes are driven by the external DDR SDRAMs during a read transaction. The data strobes are used by the memory controller to synchronize data latching. | |
| | | **State Meaning** | Asserted/Negated—Driven high when positive capture data is received and driven low when negative capture data is received. Centered in the data eye for writes; coincident with the data eye for reads. Treated as a clock. Data is valid when signals toggle. See Table 8-50 for byte lane assignments. |
| | | **Timing** | Assertion/Negation—If a READ command is registered at clock edge *n*, and the latency is programmed in TIMING_CFG_1[CASLAT] to be *m* clocks, data strobes at the DRAM assert coincident with the data on clock edge *n + m*. See the JEDEC DDR SDRAM specification for more information. |

### Table 8-3. Memory Interface Signals—Detailed Signal Descriptions (continued)

| Signal | I/O | Description | |
|---|---|---|---|
| MECC[0:7] | I/O | Error checking and correcting codes. Input and output signals for the DDR controller's bidirectional ECC bus. MECC[0:5] function in both normal and debug modes. | |
| | O | As normal mode outputs the ECC signals represent the state of ECC driven by the DDR controller on writes. As debug mode outputs MECC[0:5] provide source ID and data-valid information. See Section 8.5.11, "Error Checking and Correcting (ECC)," and Section 25.4.2.2, "Debug Information on ECC Pins," for more details. | |
| | | **State Meaning** | Asserted/Negated—Represents the state of ECC being driven by the DDR controller on writes. |
| | | **Timing** | Assertion/Negation—Same timing as MDQ<br>High impedance—Same timing as MDQ |
| | I | As inputs, the ECC signals represent the state of ECC driven by the SDRAM devices on reads. | |
| | | **State Meaning** | Asserted/Negated—Represents the state of ECC being driven by the DDR SDRAMs on reads. |
| | | **Timing** | Assertion/Negation—Same timing as MDQ<br>High impedance—Same timing as MDQ |
| MA[15:0] | O | Address bus. Memory controller outputs for the address to the DRAM. MA[15:0] carry 16 of the address bits for the DDR memory interface corresponding to the row and column address bits. MA0 is the lsb of the address output from the memory controller. | |
| | | **State Meaning** | Asserted/Negated—Represents the address driven by the DDR memory controller. Contains different portions of the address depending on the memory size and the DRAM command being issued by the memory controller. See Table 8-55 for a complete description of the mapping of these signals. |
| | | **Timing** | Assertion/Negation—The address lines are only driven when the controller has a command scheduled to issue on the address/CMD bus; otherwise they will be at high-Z. It is valid when a transaction is driven to DRAM (when $\overline{\text{MCS}}$*n* is active).<br>High impedance—When the memory controller is disabled |
| MBA[2:0] | O | Logical bank address. Outputs that drive the logical (or internal) bank address pins of the SDRAM. Each SDRAM supports four or eight addressable logical sub-banks. Bit zero of the memory controller's output bank address must be connected to bit zero of the SDRAM's input bank address. MBA0, the least-significant bit of the three bank address signals, is asserted during the mode register set command to specify the extended mode register. | |
| | | **State Meaning** | Asserted/Negated—Selects the DDR SDRAM logical (or internal) bank to be activated during the row address phase and selects the SDRAM internal bank for the read or write operation during the column address phase of the memory access. Table 8-55 describes the mapping of these signals in all cases. |
| | | **Timing** | Assertion/Negation—Same timing as MA*n*<br>High impedance—Same timing as MA*n* |

**Table 8-3. Memory Interface Signals—Detailed Signal Descriptions (continued)**

| Signal | I/O | Description | |
|---|---|---|---|
| $\overline{\text{MCAS}}$ | O | Column address strobe. Active-low SDRAM address multiplexing signal. $\overline{\text{MCAS}}$ is asserted for read or write transactions and for mode register set, refresh, and precharge commands. | |
| | | **State Meaning** | Asserted—Indicates that a valid SDRAM column address is on the address bus for read and write transactions. See Table 8-61 for more information on the states required on $\overline{\text{MCAS}}$ for various other SDRAM commands.<br>Negated—The column address is not guaranteed to be valid. |
| | | **Timing** | Assertion/Negation—Assertion and negation timing is directed by the values described in Section 8.4.1.5, “DDR SDRAM Timing Configuration 0 (TIMING_CFG_0),” Section 8.4.1.6, “DDR SDRAM Timing Configuration 1 (TIMING_CFG_1),” Section 8.4.1.7, “DDR SDRAM Timing Configuration 2 (TIMING_CFG_2),” and Section 8.4.1.4, “DDR SDRAM Timing Configuration 3 (TIMING_CFG_3).”<br>High impedance—$\overline{\text{MCAS}}$ is always driven unless the memory controller is disabled. |
| $\overline{\text{MRAS}}$ | O | Row address strobe. Active-low SDRAM address multiplexing signal. Asserted for activate commands. In addition; used for mode register set commands and refresh commands. | |
| | | **State Meaning** | Asserted—Indicates that a valid SDRAM row address is on the address bus for read and write transactions. See Table 8-61 for more information on the states required on $\overline{\text{MRAS}}$ for various other SDRAM commands.<br>Negated—The row address is not guaranteed to be valid. |
| | | **Timing** | Assertion/Negation—Assertion and negation timing is directed by the values described in Section 8.4.1.5, “DDR SDRAM Timing Configuration 0 (TIMING_CFG_0),” Section 8.4.1.6, “DDR SDRAM Timing Configuration 1 (TIMING_CFG_1),” Section 8.4.1.7, “DDR SDRAM Timing Configuration 2 (TIMING_CFG_2),” and Section 8.4.1.4, “DDR SDRAM Timing Configuration 3 (TIMING_CFG_3).”<br>High impedance—$\overline{\text{MRAS}}$ is always driven unless the memory controller is disabled. |
| $\overline{\text{MCS}}$[0:3] | O | Chip selects. Four chip selects supported by the memory controller. | |
| | | **State Meaning** | Asserted—Selects a physical SDRAM bank to perform a memory operation as described in Section 8.4.1.1, “Chip Select Memory Bounds (CSn_BNDS),” and Section 8.4.1.2, “Chip Select Configuration (CSn_CONFIG).” The DDR controller asserts one of the $\overline{\text{MCS}}$[0:3] signals to begin a memory cycle.<br>Negated—Indicates no SDRAM action during the current cycle. |
| | | **Timing** | Assertion/Negation—Asserted to signal any new transaction to the SDRAM. The transaction must adhere to the timing constraints set in TIMING_CFG_0–TIMING_CFG_3.<br>High impedance—Always driven unless the memory controller is disabled. |
| $\overline{\text{MWE}}$ | O | Write enable. Asserted when a write transaction is issued to the SDRAM. This is also used for mode registers set commands and precharge commands. | |
| | | **State Meaning** | Asserted—Indicates a memory write operation. See Table 8-61 for more information on the states required on $\overline{\text{MWE}}$ for various other SDRAM commands.<br>Negated—Indicates a memory read operation. |
| | | **Timing** | Assertion/Negation—Similar timing as $\overline{\text{MRAS}}$ and $\overline{\text{MCAS}}$. Used for write commands.<br>High impedance—$\overline{\text{MWE}}$ is always driven unless the memory controller is disabled. |

### Table 8-3. Memory Interface Signals—Detailed Signal Descriptions (continued)

| Signal | I/O | Description | |
|---|---|---|---|
| MDM[0:8] | O | DDR SDRAM data output mask. Masks unwanted bytes of data transferred during a write. They are needed to support sub-burst-size transactions (such as single-byte writes) on SDRAM where all I/O occurs in multi-byte bursts. MDM0 corresponds to the most significant byte (MSB) and MDM7 corresponds to the LSB, while MDM8 corresponds to the ECC byte. Table 8-50 shows byte lane encodings. | |
| | | **State Meaning** | Asserted—Prevents writing to DDR SDRAM. Asserted when data is written to DRAM if the corresponding byte(s) should be masked for the write. Note that the MDM*n* signals are active-high for the DDR controller. MDM*n* is part of the DDR command encoding.<br>Negated—Allows the corresponding byte to be read from or written to the SDRAM. |
| | | **Timing** | Assertion/Negation—Same timing as MDQx as outputs.<br>High impedance—Always driven unless the memory controller is disabled. |
| MODT[0:3] | O | On-Die termination. Memory controller outputs for the ODT to the DRAM. MODT[0:3] represents the on-die termination for the associated data, data masks, ECC, and data strobes. | |
| | | **State Meaning** | Asserted/Negated—Represents the ODT driven by the DDR memory controller. |
| | | **Timing** | Assertion/Negation—Driven in accordance with JEDEC DRAM specifications for on-die termination timings. It is configured through the CS*n*_CONFIG[ODT_RD_CFG] and CS*n*_CONFIG[ODT_WR_CFG] fields.<br>High impedance—Always driven. |
| MDIC[0:1] | I/O | Driver impedance calibration. Note that the MDIC signals require the use of 18.2-$\Omega$ precision 1% resistors; MDIC0 must be pulled to GND, while MDIC1 must be pulled to $GV_{DD}$. Section 8.4.1.28, "DDR Control Driver Register 2 (DDRCDR_2)," for more information on these signals. | |
| | | **State Meaning** | These pins are used for automatic calibration of the DDR IOs. |
| | | **Timing** | These are driven for four DRAM cycles at a time while the DDR controller is executing the automatic driver compensation. |
| $\overline{\text{MAPAR\_ERR}}$ | I | Address parity error. Reflects whether an address parity error has been detected by the DRAM. This signal is active low. | |
| | | **State Meaning** | Asserted—An error has been detected.<br>Negated—An error has not been detected. |
| | | **Timing** | Assertion/Negation—are driven by the registered DIMMs one DRAM cycle after the parity bit has been driven by the memory controller. This error signal should be held valid for two DRAM cycles. |
| MAPAR_OUT | O | Address parity out. Driven by the memory controller as the parity bit calculated across the address and command bits. Even parity is used, and parity is not calculated for the MCKE[0:3], MODT[0:3], or $\overline{\text{MCS}}$[0:3] signals. | |
| | | **State Meaning** | Asserted—The parity bit is high.<br>Negated—The parity bit is low. |
| | | **Timing** | Assertion/Negation—are issued one DRAM cycle after the chip select for each command. |

### 8.3.2.2 Clock Interface Signals

Table 8-4 contains the detailed descriptions of the clock signals of the DDR controller.

**Table 8-4. Clock Signals—Detailed Signal Descriptions**

| Signal | I/O | Description | |
|---|---|---|---|
| MCK[0:5],<br>$\overline{\text{MCK}}$[0:5] | O | DRAM clock outputs and their complements. See Section 8.5.4.1, "Clock Distribution." | |
| | | **State Meaning** | Asserted/Negated—The JEDEC DDR SDRAM specifications require true and complement clocks. A clock edge is seen by the SDRAM when the true and complement cross. |
| | | **Timing** | Assertion/Negation—Timing is controlled by the DDR_CLK_CNTL register at offset 0x130. |
| MCKE[0:3] | O | Clock enable. Output signals used as the clock enables to the SDRAM. MCKE[0:3] can be negated to stop clocking the DDR SDRAM. The MCKE signals should be connected to the same rank of memory as the corresponding $\overline{\text{MCS}}$ and MODT signals. For example, MCKE[0] should be connected to the same rank of memory as $\overline{\text{MCS}}$[0] and MODT[0]. | |
| | | **State Meaning** | Asserted—Clocking to the SDRAM is enabled.<br>Negated—Clocking to the SDRAM is disabled and the SDRAM should ignore signal transitions on MCK or $\overline{\text{MCK}}$. MCK/$\overline{\text{MCK}}$ are don't cares while MCKE[0:3] are negated. |
| | | **Timing** | Assertion/Negation—Asserted when DDR_SDRAM_CFG[MEM_EN] is set. Can be negated when entering dynamic power management or self refresh. Are asserted again when exiting dynamic power management or self refresh.<br>High impedance—Always driven. |

### 8.3.2.3 Debug Signals

The debug signals MSRCID[0:4] and MDVAL have no function in normal DDR controller operation. A detailed description of these signals can be found in Section 25.4.2, "DDR SDRAM Interface Debug."

## 8.4 Memory Map/Register Definition

Table 8-5 shows the register memory map for the DDR memory controller.

In this table and in the register figures and field descriptions, the following access definitions apply:

- Reserved fields are always ignored for the purposes of determining access type.
- R/W, R, and W (read/write, read only, and write only) indicate that all the non-reserved fields in a register have the same access type.
- w1c indicates that all of the non-reserved fields in a register are cleared by writing ones to them.
- Mixed indicates a combination of access types.
- Special is used when no other category applies. In this case the register figure and field description table should be read carefully.

**Table 8-5. DDR Memory Controller Memory Map**

| Offset | Register | Access | Reset | Section/Page |
|---|---|---|---|---|
| **DDR Memory Controller—Block Base Address 0x0_2000** | | | | |
| 0x000 | CS0_BNDS—Chip select 0 memory bounds | R/W | 0x0000_0000 | 8.4.1.1/8-12 |
| 0x008 | CS1_BNDS—Chip select 1 memory bounds | R/W | 0x0000_0000 | 8.4.1.1/8-12 |

Table 8-5. DDR Memory Controller Memory Map (continued)

| Offset | Register | Access | Reset | Section/Page |
|---|---|---|---|---|
| 0x010 | CS2_BNDS—Chip select 2 memory bounds | R/W | 0x0000_0000 | 8.4.1.1/8-12 |
| 0x018 | CS3_BNDS—Chip select 3 memory bounds | R/W | 0x0000_0000 | 8.4.1.1/8-12 |
| 0x080 | CS0_CONFIG—Chip select 0 configuration | R/W | 0x0000_0000 | 8.4.1.2/8-13 |
| 0x084 | CS1_CONFIG—Chip select 1 configuration | R/W | 0x0000_0000 | 8.4.1.2/8-13 |
| 0x088 | CS2_CONFIG—Chip select 2 configuration | R/W | 0x0000_0000 | 8.4.1.2/8-13 |
| 0x08C | CS3_CONFIG—Chip select 3 configuration | R/W | 0x0000_0000 | 8.4.1.2/8-13 |
| 0x0C0 | CS0_CONFIG_2—Chip select 0 configuration 2 | R/W | 0x0000_0000 | 8.4.1.3/8-15 |
| 0x0C4 | CS1_CONFIG_2—Chip select 1 configuration 2 | R/W | 0x0000_0000 | 8.4.1.3/8-15 |
| 0x0C8 | CS2_CONFIG_2—Chip select 2 configuration 2 | R/W | 0x0000_0000 | 8.4.1.3/8-15 |
| 0x0CC | CS3_CONFIG_2—Chip select 3 configuration 2 | R/W | 0x0000_0000 | 8.4.1.3/8-15 |
| 0x100 | TIMING_CFG_3—DDR SDRAM timing configuration 3 | R/W | 0x0000_0000 | 8.4.1.4/8-16 |
| 0x104 | TIMING_CFG_0—DDR SDRAM timing configuration 0 | R/W | 0x0011_0105 | 8.4.1.5/8-17 |
| 0x108 | TIMING_CFG_1—DDR SDRAM timing configuration 1 | R/W | 0x0000_0000 | 8.4.1.6/8-19 |
| 0x10C | TIMING_CFG_2—DDR SDRAM timing configuration 2 | R/W | 0x0000_0000 | 8.4.1.7/8-21 |
| 0x110 | DDR_SDRAM_CFG—DDR SDRAM control configuration | R/W | 0x0200_0000 | 8.4.1.8/8-23 |
| 0x114 | DDR_SDRAM_CFG_2—DDR SDRAM control configuration 2 | R/W | 0x0000_0000 | 8.4.1.9/8-26 |
| 0x118 | DDR_SDRAM_MODE—DDR SDRAM mode configuration | R/W | 0x0000_0000 | 8.4.1.10/8-29 |
| 0x11C | DDR_SDRAM_MODE_2—DDR SDRAM mode configuration 2 | R/W | 0x0000_0000 | 8.4.1.11/8-29 |
| 0x120 | DDR_SDRAM_MD_CNTL—DDR SDRAM mode control | R/W | 0x0000_0000 | 8.4.1.12/8-30 |
| 0x124 | DDR_SDRAM_INTERVAL—DDR SDRAM interval configuration | R/W | 0x0000_0000 | 8.4.1.13/8-33 |
| 0x128 | DDR_DATA_INIT—DDR SDRAM data initialization | R/W | 0x0000_0000 | 8.4.1.14/8-33 |
| 0x130 | DDR_SDRAM_CLK_CNTL—DDR SDRAM clock control | R/W | 0x0200_0000 | 8.4.1.15/8-34 |
| 0x140–<br>0x144 | Reserved | — | — | — |
| 0x148 | DDR_INIT_ADDR—DDR training initialization address | R/W | 0x0000_0000 | 8.4.1.16/8-34 |
| 0x14C | DDR_INIT_EXT_ADDR—DDR training initialization extended address | R/W | 0x0000_0000 | 8.4.1.17/8-35 |
| 0x150–<br>0x15F | Reserved | — | — | — |
| 0x160 | TIMING_CFG_4— DDR SDRAM timing configuration 4 | R/W | 0x0000_0000 | 8.4.1.18/8-36 |
| 0x164 | TIMING_CFG_5— DDR SDRAM timing configuration 5 | R/W | 0x0000_0000 | 8.4.1.19/8-37 |
| 0x168–<br>0x16F | Reserved | — | — | — |
| 0x170 | DDR_ZQ_CNTL— DDR ZQ calibration control | R/W | 0x0000_0000 | 8.4.1.20/8-39 |
| 0x174 | DDR_WRLVL_CNTL— DDR write leveling control | R/W | 0x0000_0000 | 8.4.1.21/8-40 |
| 0x178 | Reserved | — | — | — |
| 0x17C | DDR_SR_CNTR — DDR Self Refresh Counter | R/W | 0x0000_0000 | 8.4.1.22/8-43 |
| 0x180 | DDR_SDRAM_RCW_1 — DDR Register Control Words 1 | R/W | 0x0000_0000 | 8.4.1.23/8-44 |
| 0x184 | DDR_SDRAM_RCW_2 — DDR Register Control Words 2 | R/W | 0x0000_0000 | 8.4.1.24/8-45 |

Table 8-5. DDR Memory Controller Memory Map (continued)

| Offset | Register | Access | Reset | Section/Page |
|---|---|---|---|---|
| 0x188–0xB1F | Reserved | — | — | — |
| 0xB20 | DDRDSR_1—DDR Debug Status Register 1 | R | 0x0000_0000 | 8.4.1.25/8-46 |
| 0xB24 | DDRDSR_2—DDR Debug Status Register 2 | R | 0x0000_0000 | 8.4.1.26/8-47 |
| 0xB28 | DDRCDR_1—DDR Control Driver Register 1 | R/W | 0x0000_0000 | 8.4.1.27/8-47 |
| 0xB2C | DDRCDR_2—DDR Control Driver Register 2 | R/W | 0x0000_0000 | 8.4.1.28/8-50 |
| 0xB30–0xBF7 | Reserved | — | — | — |
| 0xBF8 | DDR_IP_REV1—DDR IP block revision 1 | R | 0x*nnnn_nnnn*[1] | 8.4.1.29/8-50 |
| 0xBFC | DDR_IP_REV2—DDR IP block revision 2 | R | 0x00*nn*_00*nn*[1] | 8.4.1.30/8-51 |
| 0xE00 | DATA_ERR_INJECT_HI—Memory data path error injection mask high | R/W | 0x0000_0000 | 8.4.1.31/8-51 |
| 0xE04 | DATA_ERR_INJECT_LO—Memory data path error injection mask low | R/W | 0x0000_0000 | 8.4.1.32/8-52 |
| 0xE08 | ERR_INJECT—Memory data path error injection mask ECC | R/W | 0x0000_0000 | 8.4.1.33/8-52 |
| 0xE20 | CAPTURE_DATA_HI—Memory data path read capture high | R/W | 0x0000_0000 | 8.4.1.34/8-53 |
| 0xE24 | CAPTURE_DATA_LO—Memory data path read capture low | R/W | 0x0000_0000 | 8.4.1.35/8-54 |
| 0xE28 | CAPTURE_ECC—Memory data path read capture ECC | R/W | 0x0000_0000 | 8.4.1.36/8-54 |
| 0xE40 | ERR_DETECT—Memory error detect | w1c | 0x0000_0000 | 8.4.1.37/8-54 |
| 0xE44 | ERR_DISABLE—Memory error disable | R/W | 0x0000_0000 | 8.4.1.38/8-56 |
| 0xE48 | ERR_INT_EN—Memory error interrupt enable | R/W | 0x0000_0000 | 8.4.1.39/8-57 |
| 0xE4C | CAPTURE_ATTRIBUTES—Memory error attributes capture | R/W | 0x0000_0000 | 8.4.1.40/8-58 |
| 0xE50 | CAPTURE_ADDRESS—Memory error address capture | R/W | 0x0000_0000 | 8.4.1.41/8-58 |
| 0xE54 | CAPTURE_EXT_ADDRESS—Memory error extended address capture | R/W | 0x0000_0000 | 8.4.1.42/8-59 |
| 0xE58 | ERR_SBE—Single-Bit ECC memory error management | R/W | 0x0000_0000 | 8.4.1.43/8-59 |

[1] Implementation-dependent reset values are listed in specified section/page.

## 8.4.1 Register Descriptions

This section describes the DDR memory controller registers. Shading indicates reserved fields that should not be written.

### 8.4.1.1 Chip Select Memory Bounds (CS*n*_BNDS)

The chip select bounds registers (CS*n*_BNDS) define the starting and ending address of the memory space that corresponds to the individual chip selects. Note that the size specified in CS*n*_BNDS should equal the size of physical DRAM. Also, note that EA*n* must be greater than or equal to SA*n*.

If chip select interleaving is enabled, all fields in the lower interleaved chip select are used, and the other chip selects' bounds registers are unused. For example, if chip selects 0 and 1 are interleaved, all fields in CS0_BNDS are used, and all fields in CS1_BNDS are unused.

CS*n*_BNDS are shown in Figure 8-2.

Offset 0x000, 0x008, 0x010, 0x018 Access: Read/Write

| | 0–3 | 4–15 | 16–19 | 20–31 |
|---|---|---|---|---|
| R<br>W | — | SA*n* | — | EA*n* |
| Reset | All zeros | | | |

**Figure 8-2. Chip Select Bounds Registers (CS*n*_BNDS)**

Table 8-6 describes the CS*n*_BNDS register fields.

**Table 8-6. CS*n*_BNDS Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–3 | — | Reserved |
| 4–15 | SA*n* | Starting address for chip select (bank) *n.* This value is compared against the 12 msbs of the 36-bit address. |
| 16–19 | — | Reserved |
| 20–31 | EA*n* | Ending address for chip select (bank) *n.* This value is compared against the 12 msbs of the 36-bit address. |

### 8.4.1.2 Chip Select Configuration (CS*n*_CONFIG)

The chip select configuration (CS*n*_CONFIG) registers shown in Figure 8-3 enable the DDR chip selects and set the number of row and column bits used for each chip select. These registers should be loaded with the correct number of row and column bits for each SDRAM. Because CS*n*_CONFIG[ROW_BITS_CS_*n*, COL_BITS_CS_*n*] establish address multiplexing, the user should take great care to set these values correctly.

If chip select interleaving is enabled, then all fields in the lower interleaved chip select are used, and the other registers' fields are unused, with the exception of the ODT_RD_CFG and ODT_WR_CFG fields. For example, if chip selects 0 and 1 are interleaved, all fields in CS0_CONFIG are used, but only the ODT_RD_CFG and ODT_WR_CFG fields in CS1_CONFIG are used.

Offset 0x080, 0x084, 0x088, 0x08C Access: Read/Write

| | 0 | 1–7 | 8 | 9–11 | 12 | 13–15 |
|---|---|---|---|---|---|---|
| R<br>W | CS_*n* _EN | — | AP_*n*_EN | ODT_RD_CFG | — | ODT_WR_CFG |
| Reset | All zeros | | | | | |

| | 16–17 | 18–20 | 21–23 | 24–28 | 29–31 |
|---|---|---|---|---|---|
| R<br>W | BA_BITS_CS_*n* | — | ROW_BITS_CS_*n* | — | COL_BITS_CS_*n* |
| Reset | All zeros | | | | |

**Figure 8-3. Chip Select Configuration Register (CS*n*_CONFIG)**

Table 8-7 describes the CS*n*_CONFIG register fields.

**Table 8-7. CS*n*_CONFIG Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | CS_*n*_EN | Chip select *n* enable<br>0 Chip select *n* is not active<br>1 Chip select *n* is active and assumes the state set in CS*n*_BNDS. |
| 1–7 | — | Reserved |
| 8 | AP_*n*_EN | Chip select *n* auto-precharge enable<br>0 Chip select *n* is only auto-precharged if global auto-precharge mode is enabled (DDR_SDRAM_INTERVAL[BSTOPRE] = 0).<br>1 Chip select *n* always issues an auto-precharge for read and write transactions. |
| 9–11 | ODT_RD_CFG | ODT for reads configuration. Note that CAS latency plus additive latency must be at least 3 cycles for ODT_RD_CFG to be enabled. ODT should only be used with DDR2 or DDR3 memories.<br>000 Never assert ODT for reads<br>001 Assert ODT only during reads to CS*n*<br>010 Assert ODT only during reads to other chip selects<br>011 Assert ODT only during reads to other DIMM modules. It is assumed that CS0 and CS1 are on the same DIMM module, whereas CS2 and CS3 are on a separate DIMM module.<br>100 Assert ODT for all reads<br>101–111 Reserved |
| 12 | — | Reserved |
| 13–15 | ODT_WR_CFG | ODT for writes configuration. Note that write latency plus additive latency must be at least 3 cycles for ODT _WR_CFG to be enabled. ODT should only be used with DDR2 or DDR3 memories.<br>000 Never assert ODT for writes<br>001 Assert ODT only during writes to CS*n*<br>010 Assert ODT only during writes to other chip selects<br>011 Assert ODT only during writes to other DIMM modules. It is assumed that CS0 and CS1 are on the same DIMM module, whereas CS2 and CS3 are on a separate DIMM module.<br>100 Assert ODT for all writes<br>101–111 Reserved |
| 16–17 | BA_BITS_CS_*n* | Number of bank bits for SDRAM on chip select *n*. These bits correspond to the sub-bank bits driven on MBA*n* in Table 8-55and Table 8-55.<br>00 2 logical bank bits<br>01 3 logical bank bits<br>10–11 Reserved |
| 18–20 | — | Reserved |
| 21–23 | ROW_BITS_CS_*n* | Number of row bits for SDRAM on chip select *n*. See Table 8-55and Table 8-55 for details.<br>000 12 row bits<br>001 13 row bits<br>010 14 row bits<br>011 15 row bits<br>100 16 row bits<br>101–111 Reserved |

Table 8-7. CS*n*_CONFIG Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 24–28 | — | Reserved |
| 29–31 | COL_BITS_CS_*n* | Number of column bits for SDRAM on chip select *n.* For DDR, the decoding is as follows:<br>000 8 column bits<br>001 9 column bits<br>010 10 column bits<br>011 11 column bits<br>100–111 Reserved |

### 8.4.1.3 Chip Select Configuration 2 (CS*n*_CONFIG_2)

The chip select configuration (CS*n*_CONFIG_2) registers shown in Figure 8-4 enable the partial array self refresh address decode in each chip select.

If chip select interleaving is enabled, then all fields in the lower interleaved chip select are used, and the other registers' fields are unused.

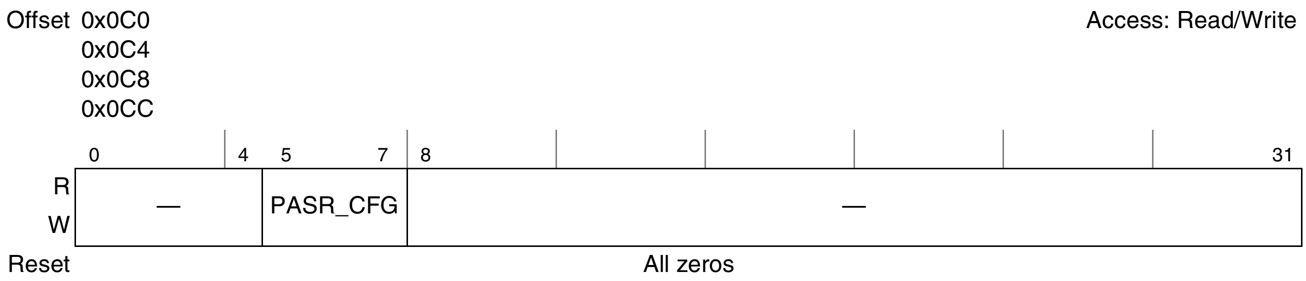


Figure 8-4. Chip Select Configuration Register 2 (CS*n*_CONFIG_2)

Table 8-8 describes the CS*n*_CONFIG_2 register fields.

Table 8-8. CS*n*_CONFIG_2 Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–4 | — | Reserved |
| 5–7 | PASR_CFG | Partial array self refresh config. Controls the bits that are placed on MA[2:0] during the write to the EMRS(2) register when the automatic hardware DRAM initialization is used (DDR_SDRAM_CFG[BI] is cleared when DDR_SDRAM_CFG[MEM_EN] is set). If this field is a non-zero value, then it overrides the least significant 3 bits in DDR_SDRAM_MODE_2[ESDMODE2] during the automatic initialization for chip select *n*. In addition, if a non-zero value is programmed in this field, then the address decode for chip select *n* is optimized for partial array self refresh, as shown in Section 8.5.2, "DDR SDRAM Address Multiplexing.<br>000 Partial array self refresh is disabled<br>001–111Partial array self refresh is enabled per JEDEC specifications. Overriding the least significant 3 bits of EMRS or EMRS(2) is only supported for DDR2 and DDR3 memory types. |
| 8–31 | — | Reserved |

### 8.4.1.4 DDR SDRAM Timing Configuration 3 (TIMING_CFG_3)

DDR SDRAM timing configuration register 3, shown in Figure 8-5, sets the extended refresh recovery time, which is combined with TIMING_CFG_1[REFREC] to determine the full refresh recovery time.

Offset 0x100 Access: Read/Write

| | 0–6 | 7 | 8–11 | 12–15 | 16–18 | 19 | 20–28 | 29–31 |
|---|---|---|---|---|---|---|---|---|
| R<br>W | — | EXT_ACTTOPRE | — | EXT_REFREC | — | EXT_CASLAT | — | CNTL_ADJ |
| Reset | All zeros | | | | | | | |

**Figure 8-5. DDR SDRAM Timing Configuration 3 (TIMING_CFG_3)**

Table 8-9 describes TIMING_CFG_3 fields.

**Table 8-9. TIMING_CFG_3 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–6 | — | Reserved, should be cleared. |
| 7 | EXT_ACTTOPRE | Extended Activate to precharge interval ($t_{RAS}$). Determines the number of clock cycles from an activate command until a precharge command is allowed. This field is concatenated with TIMING_CFG_1[ACTTOPRE] to obtain a 5-bit value for the total activate to precharge. Note that a 5-bit value of 0_0000 is the same as a 5-bit value of 1_0000. Both values represent 16 cycles.<br>0 0 clocks<br>1 16 clocks |
| 8–11 | — | Reserved, should be cleared. |
| 12–15 | EXT_REFREC | Extended refresh recovery time ($t_{RFC}$). Controls the number of clock cycles from a refresh command until an activate command is allowed. This field is concatenated with TIMING_CFG_1[REFREC] to obtain an 8-bit value for the total refresh recovery. Note that hardware adds an additional 8 clock cycles to the final, 8-bit value of the refresh recovery. $t_{RFC}$ = {EXT_REFREC ll REFREC} + 8, such that $t_{RFC}$ is calculated as follows:<br>0000 0 clocks<br>0001 16 clocks<br>0010 32 clocks<br>0011 48 clocks<br>0100 64 clocks<br>0101 80 clocks<br>0110 96 clocks<br>0111 112 clocks<br>1000 128 clocks<br>1001 144 clocks<br>1010 160 clocks<br>1011 176 clocks<br>1100 192 clocks<br>1101 208 clocks<br>1110 224 clocks<br>1111 240 clocks |
| 16–18 | — | Reserved, should be cleared. |

**Table 8-9. TIMING_CFG_3 Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 19 | EXT_CASLAT | Extended $\overline{\text{MCAS}}$ latency from READ command. Number of clock cycles between registration of a READ command by the SDRAM and the availability of the first output data. If a READ command is registered at clock edge $n$ and the latency is $m$ clocks, data is available nominally coincident with clock edge $n + m$. This field is concatenated with TIMING_CFG_1[CASLAT] to obtain a 5-bit value for the total CAS latency. Note that if this bit is set, then 8 clocks are added to the programmed value in TIMING_CFG_1[CASLAT].<br>0 0 clocks 1 8 clocks |
| 20–28 | — | Reserved, should be cleared. |
| 29–31 | CNTL_ADJ | Control Adjust. Controls the amount of delay to add to the lightly loaded control signals w/ respect to all other DRAM address and command signals. The signals affected by this field are MODT[0:3], $\overline{\text{MCS}}$[0:3], and MCKE[0:3]<br>000 MODT[0:3], $\overline{\text{MCS}}$[0:3], and MCKE[0:3] are launched aligned with the other DRAM address and control signals.<br>001 MODT[0:3], $\overline{\text{MCS}}$[0:3], and MCKE[0:3] are launched 1/2 platform cycle later than the other DRAM address and control signals.<br>010 MODT[0:3], $\overline{\text{MCS}}$[0:3], and MCKE[0:3] are launched 1 platform cycle later than the other DRAM address and control signals.<br>011 MODT[0:3], $\overline{\text{MCS}}$[0:3], and MCKE[0:3] are launched 3/2 platform cycles later than the other DRAM address and control signals.<br>100 MODT[0:3], $\overline{\text{MCS}}$[0:3], and MCKE[0:3] are launched 2 platform cycles later than the other DRAM address and control signals.<br>101 MODT[0:3], $\overline{\text{MCS}}$[0:3], and MCKE[0:3] are launched 5/2 platform cycles later than the other DRAM address and control signals.<br>110-111Reserved |

### 8.4.1.5 DDR SDRAM Timing Configuration 0 (TIMING_CFG_0)

DDR SDRAM timing configuration register 0, shown in Figure 8-6, sets the number of clock cycles between various SDRAM control commands.

Offset 0x104 Access: Read/Write

| | 0 1 | 2 3 | 4 5 | 6 7 | 8 | 9 11 | 12 15 | 16 19 | 20 23 | 24 27 | 28 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | RWT | WRT | RRT | WWT | — | ACT_PD_EXIT | PRE_PD_EXIT | — | ODT_PD_EXIT | — | MRS_CYC |
| Reset | 0 0 | 0 0 | 0 0 | 0 0 | 0 | 0 0 1 | 0 0 0 1 | 0 0 0 0 | 0 0 0 1 | 0 0 0 0 | 0 1 0 1 |

**Figure 8-6. DDR SDRAM Timing Configuration 0 (TIMING_CFG_0)**

Table 8-10 describes TIMING_CFG_0 fields.

**Table 8-10. TIMING_CFG_0 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–1 | RWT | Read-to-write turnaround ($t_{RTW}$). Specifies how many extra cycles are added between a read to write turnaround. If 0 clocks is chosen, then the DDR controller uses a fixed number based on the CAS latency and write latency. Choosing a value other than 0 adds extra cycles past this default calculation. As a default the DDR controller determines the read-to-write turnaround as CL – WL + BL/2 + 2. In this equation, CL is the CAS latency rounded up to the next integer, WL is the programmed write latency, and BL is the burst length.<br>00 0 clocks<br>01 1 clock<br>10 2 clocks<br>11 3 clocks |
| 2–3 | WRT | Write-to-read turnaround. Specifies how many extra cycles are added between a write to read turnaround. If 0 clocks is chosen, then the DDR controller uses a fixed number based on the, read latency, and write latency. Choosing a value other than 0 adds extra cycles past this default calculation. As a default, the DDR controller determines the write-to-read turnaround as WL – CL + BL/2 + 1. In this equation, CL is the CAS latency rounded down to the next integer, WL is the programmed write latency, and BL is the burst length.<br>00 0 clocks<br>01 1 clock<br>10 2 clocks<br>11 3 clocks |
| 4–5 | RRT | Read-to-read turnaround. Specifies how many extra cycles are added between reads to different chip selects. As a default, 3 cycles are required between read commands to different chip selects. Extra cycles may be added with this field. Note: If 8-beat bursts are enabled, then 5 cycles are the default. Note that DDR2 does not support 8-beat bursts.<br>00 0 clocks<br>01 1 clock<br>10 2 clocks<br>11 3 clocks |
| 6–7 | WWT | Write-to-write turnaround. Specifies how many extra cycles are added between writes to different chip selects. As a default, 2 cycles are required between write commands to different chip selects. Extra cycles may be added with this field. Note: If 8-beat bursts are enabled, then 4 cycles are the default. Note that DDR2 does not support 8-beat bursts.<br>00 0 clocks<br>01 1 clock<br>10 2 clocks<br>11 3 clocks |
| 8 | — | Reserved, should be cleared. |
| 9–11 | ACT_PD_EXIT | Active powerdown exit timing ($t_{XARD}$ and $t_{XARDS}$). Specifies how many clock cycles to wait after exiting active powerdown before issuing any command.<br>000 Reserved<br>001 1 clock<br>010 2 clocks<br>011 3 clocks<br>100 4 clocks<br>101 5 clocks<br>110 6 clocks<br>111 7 clocks |
| 12–15 | PRE_PD_EXIT | Precharge powerdown exit timing ($t_{XP}$). Specifies how many clock cycles to wait after exiting precharge powerdown before issuing any command.<br>0000 Reserved<br>0001 1 clock<br>0010 2 clocks<br>0011 3 clocks<br>0100 4 clocks<br>0101 5 clocks<br>0110 6 clocks<br>0111 7 clocks<br>1000 8 clocks<br>1001 9 clocks<br>1010 10 clocks<br>1011 11 clocks<br>1100 12 clocks<br>1101 13 clocks<br>1110 14 clocks<br>1111 15 clocks |
| 16–19 | — | Reserved, should be cleared. |

**Table 8-10. TIMING_CFG_0 Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 20–23 | ODT_PD_EXIT | ODT powerdown exit timing ($t_{AXPD}$). Specifies how many clocks must pass after exiting powerdown before ODT may be asserted.<br>0000 0 clock<br>0001 1 clock<br>0010 2 clocks<br>0011 3 clocks<br>0100 4 clocks<br>0101 5 clocks<br>0110 6 clocks<br>0111 7 clocks<br>1000 8 clocks<br>1001 9 clocks<br>1010 10 clocks<br>1011 11 clocks<br>1100 12 clocks<br>1101 13 clocks<br>1110 14 clocks<br>1111 15 clocks |
| 24–27 | — | Reserved, should be cleared. |
| 28–31 | MRS_CYC | Mode register set cycle time ($t_{MRD}$). Specifies the number of cycles that must pass after a Mode Register Set command until any other command.<br>0000 Reserved<br>0001 1 clock<br>0010 2 clocks<br>0011 3 clocks<br>0100 4 clocks<br>0101 5 clocks<br>0110 6 clocks<br>0111 7 clocks<br>1000 8 clocks<br>1001 9 clocks<br>1010 10 clocks<br>1011 11 clocks<br>1100 12 clocks<br>1101 13 clocks<br>1110 14 clocks<br>1111 15 clocks |

### 8.4.1.6 DDR SDRAM Timing Configuration 1 (TIMING_CFG_1)

DDR SDRAM timing configuration register 1, shown in Figure 8-7, sets the number of clock cycles between various SDRAM control commands.

Offset 0x108 Access: Read/Write

| | 0 3 | 4 7 | 8 11 | 12 15 | 16 19 | 20 23 | 24 | 25 27 | 28 | 29 31 |
|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | PRETOACT | ACTTOPRE | ACTTORW | CASLAT | REFREC | WRREC | — | ACTTOACT | — | WRTORD |
| Reset | All zeros | | | | | | | | | |

**Figure 8-7. DDR SDRAM Timing Configuration 1 (TIMING_CFG_1)**

Table 8-11 describes TIMING_CFG_1 fields.

**Table 8-11. TIMING_CFG_1 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–3 | PRETOACT | Precharge-to-activate interval ($t_{RP}$). Determines the number of clock cycles from a precharge command until an activate or refresh command is allowed.<br>0000 Reserved<br>0001 1 clock<br>0010 2 clocks<br>0011 3 clocks<br>0100 4 clocks<br>0101 5 clocks<br>0110 6 clocks<br>0111 7 clocks<br>1000 8 clocks<br>1001 9 clocks<br>1010 10 clocks<br>1011 11 clocks<br>1100 12 clocks<br>1101 13 clocks<br>1110 14 clocks<br>1111 15 clocks |
| 4–7 | ACTTOPRE | Activate to precharge interval ($t_{RAS}$). Determines the number of clock cycles from an activate command until a precharge command is allowed. This field is concatenated with TIMING_CFG_3[EXT_ACTTOPRE] to obtain a 5-bit value for the total activate to precharge time. Note that the decode of 0000–0011 is equal to 16-19 clocks when TIMING_CFG_3[EXT_ACTTOPRE] = 0, but it is equal to 0-3 clocks when TIMING_CFG_3[EXT_ACTTOPRE] = 1.<br>0000 16 clocks<br>0001 17 clocks<br>0010 18 clocks<br>0011 19 clocks<br>0100 4 clocks<br>0101 5 clocks<br>0110 6 clocks<br>0111 7 clocks<br>...<br>1111 15 clocks |
| 8–11 | ACTTORW | Activate to read/write interval for SDRAM ($t_{RCD}$). Controls the number of clock cycles from an activate command until a read or write command is allowed.<br>0000 Reserved<br>0001 1 clock<br>0010 2 clocks<br>0011 3 clocks<br>0100 4 clocks<br>0101 5 clocks<br>0110 6 clocks<br>0111 7 clocks<br>1000 8 clocks<br>1001 9 clocks<br>1010 10 clocks<br>1011 11 clocks<br>1100 12 clocks<br>1101 13 clocks<br>1110 14 clocks<br>1111 15 clocks |
| 12–15 | CASLAT | $\overline{\text{MCAS}}$ latency from READ command. Number of clock cycles between registration of a READ command by the SDRAM and the availability of the first output data. If a READ command is registered at clock edge $n$ and the latency is $m$ clocks, data is available nominally coincident with clock edge $n + m$. This field is concatenated with TIMING_CFG_3[EXT_CASLAT] to obtain a 5-bit value for the total CAS latency. This value must be programmed at initialization as described in Section 8.4.1.9, "DDR SDRAM Control Configuration 2 (DDR_SDRAM_CFG_2).")<br>0000 Reserved<br>0001 1 clock<br>0010 1.5 clocks<br>0011 2 clocks<br>0100 2.5 clocks<br>0101 3 clocks<br>0110 3.5 clocks<br>0111 4 clocks<br>1000 4.5 clocks<br>1001 5 clocks<br>1010 5.5 clocks<br>1011 6 clocks<br>1100 6.5 clocks<br>1101 7 clocks<br>1110 7.5 clocks<br>1111 8 clocks |

Table 8-11. TIMING_CFG_1 Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 16–19 | REFREC | Refresh recovery time ($t_{RFC}$). Controls the number of clock cycles from a refresh command until an activate command is allowed. This field is concatenated with TIMING_CFG_3[EXTREFREC] to obtain a 7-bit value for the total refresh recovery. Note that hardware adds an additional 8 clock cycles to the final, 7-bit value of the refresh recovery, such that $t_{RFC}$ is calculated as follows: $t_{RFC}$ = {EXT_REFREC \|\| REFREC} + 8.<br>0000 8 clocks<br>0001 9 clocks<br>0010 10 clocks<br>0011 11 clocks<br>...<br>1111 23 clocks |
| 20–23 | WRREC | Last data to precharge minimum interval ($t_{WR}$). Determines the number of clock cycles from the last data associated with a write command until a precharge command is allowed. If DDR_SDRAM_CFG_2[OBC_CFG] is set, then this field needs to be programmed to *($t_{WR}$ + 2* cycles).<br>0000 Reserved<br>0001 1 clock<br>0010 2 clocks<br>0011 3 clocks<br>0100 4 clocks<br>0101 5 clocks<br>0110 6 clocks<br>0111 7 clocks<br>1000 8 clocks<br>1001 9 clocks<br>1010 10 clocks<br>1011 11 clocks<br>1100 12 clocks<br>1101 13 clocks<br>1110 14 clocks<br>1111 15 clocks |
| 24 | — | Reserved, should be cleared. |
| 25–27 | ACTTOACT | Activate-to-activate interval ($t_{RRD}$). Number of clock cycles from an activate command until another activate command is allowed for a different logical bank in the same physical bank (chip select).<br>000 Reserved<br>001 1 clock<br>010 2 clocks<br>011 3 clocks<br>100 4 clocks<br>101 5 clocks<br>110 6 clocks<br>111 7 clocks |
| 28 | — | Reserved, should be cleared. |
| 29–31 | WRTORD | Last write data pair to read command issue interval ($t_{WTR}$). Number of clock cycles between the last write data pair and the subsequent read command to the same physical bank. If DDR_SDRAM_CFG_2[OBC_CFG] is set, then this field needs to be programmed to *($t_{WTR}$ + 2* cycles).<br>000 Reserved<br>001 1 clock<br>010 2 clocks<br>011 3 clocks<br>100 4 clocks<br>101 5 clocks<br>110 6 clocks<br>111 7 clocks |

### 8.4.1.7 DDR SDRAM Timing Configuration 2 (TIMING_CFG_2)

DDR SDRAM timing configuration 2, shown in Figure 8-8, sets the clock delay to data for writes.

Offset 0x10C    Access: Read/Write

| | 0–3 | 4–8 | 9–12 | 13–15 | 16–18 | 19–21 | 22 | 23–25 | 26–31 |
|---|---|---|---|---|---|---|---|---|---|
| R<br>W | ADD_LAT | CPO | WR_LAT | — | RD_TO_PRE | WR_DATA_DELAY | — | CKE_PLS | FOUR_ACT |
| Reset | All zeros | | | | | | | | |

Figure 8-8. DDR SDRAM Timing Configuration 2 Register (TIMING_CFG_2)

Table 8-12 describes the TIMING_CFG_2 fields.

**Table 8-12. TIMING_CFG_2 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–3 | ADD_LAT | Additive latency. The additive latency must be set to a value less than TIMING_CFG_1[ACTTORW]. (DDR2-specific)<br>0000 0 clocks<br>0001 1 clock<br>0010 2 clocks<br>0011 3 clocks<br>0100 4 clocks<br>0101 5 clocks<br>0110 6 clocks<br>0111 7 clocks<br>1000 8 clocks<br>1001 9 clocks<br>1010 10 clocks<br>1011 11 clocks<br>1100 12 clocks<br>1101 13 clocks<br>1110 14 clocks<br>1111 15 clocks |
| 4–8 | CPO[1] | $\overline{\text{MCAS}}$-to-preamble override. Defines the number of DRAM cycles between when a read is issued and when the corresponding DQS preamble is valid for the memory controller. For these decodings, “READ_LAT” is equal to the CAS latency plus the additive latency.<br>00000 READ_LAT + 1<br>00001 Reserved<br>00010 READ_LAT<br>00011 READ_LAT + 1/4<br>00100 READ_LAT + 1/2<br>00101 READ_LAT + 3/4<br>00110 READ_LAT + 1<br>00111 READ_LAT + 5/4<br>01000 READ_LAT + 3/2<br>01001 READ_LAT + 7/4<br>01010 READ_LAT + 2<br>01011 READ_LAT + 9/4<br>01100 READ_LAT + 5/2<br>01101 READ_LAT + 11/4<br>01110 READ_LAT + 3<br>01111 READ_LAT + 13/4<br>10000 READ_LAT + 7/2<br>10001 READ_LAT + 15/4<br>10010 READ_LAT + 4<br>10011 READ_LAT + 17/4<br>10100 READ_LAT + 9/2<br>10101 READ_LAT + 19/4<br>10110 READ_LAT + 5<br>10111 READ_LAT + 21/4<br>11000 READ_LAT + 11/2<br>11001 READ_LAT + 23/4<br>11010 READ_LAT + 6<br>11011 READ_LAT + 25/4<br>11100 READ_LAT +13/2<br>11101 READ_LAT + 27/4<br>11110 READ_LAT + 7<br>11111 Reserved |
| 9–12 | WR_LAT | Write latency. Note that the total write latency for DDR2 is equal to WR_LAT + ADD_LAT; the write latency for DDR1 is 1. DDR1 is not supported for the MPC8536E. If a write latency of 1 is desired, then the additive latency must also be set to at least 1 cycle.<br>0000 Reserved<br>0001 1 clock<br>0010 2 clocks<br>0011 3 clocks<br>0100 4 clocks<br>0101 5 clocks<br>0110 6 clocks<br>0111 7 clocks<br>1000 8 clocks<br>1001 9 clocks<br>1010 10 clocks<br>1011 11 clocks<br>1100 12 clocks<br>1101 13 clocks<br>1110 14 clocks<br>1111 15 clocks |
| 13–15 | — | Reserved |
| 16–18 | RD_TO_PRE | Read to precharge ($t_{RTP}$). For DDR2, with a non-zero ADD_LAT value, takes a minimum of ADD_LAT + $t_{RTP}$ cycles between read and precharge. If DDR_SDRAM_CFG_2[OBC_CFG] is set, then this field needs to be programmed to *($t_{RTP}$ + 2* cycles)<br>000 Reserved<br>001 1 cycle<br>010 2 cycles<br>011 3 cycles<br>100 4 cycles<br>101 5 cycles<br>110 6 cycles<br>111 7 cycles |

Table 8-12. TIMING_CFG_2 Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 19–21 | WR_DATA_DELAY | Write command to write data strobe timing adjustment. Controls the amount of delay applied to the data and data strobes for writes. See Section 8.5.7, “DDR SDRAM Write Timing Adjustments,” for details. The write preamble typically is driven high for 1/2 DRAM cycle, and then it is driven low for 1/2 DRAM cycle. However, for WR_DATA_DELAY settings of 0 clocks and 1/4 clocks, the write preamble is driven low for the entire DRAM cycle. If the preamble needs to switch high first (to meet DDR3 specifications), then these values should not be used.<br>000 0 clock delay<br>001 1/4 clock delay<br>010 1/2 clock delay<br>011 3/4 clock delay<br>100 1 clock delay<br>101 5/4 clock delay<br>110 3/2 clock delay<br>111 Reserved |
| 22 | — | Reserved |
| 23–25 | CKE_PLS | Minimum CKE pulse width ($t_{CKE}$).<br>000 Reserved<br>001 1 clock<br>010 2 clocks<br>011 3 clocks<br>100 4 clocks<br>101 5 clocks<br>110 6 clocks<br>111 7 clocks |
| 26–31 | FOUR_ACT | Window for four activates ($t_{FAW}$). This is applied to DDR2/DDR3 with eight logical banks only.<br>000000 Reserved<br>000001 1 cycle<br>000010 2 cycles<br>000011 3 cycles<br>000100 4 cycles<br>...<br>011110 30 cycles<br>011111 31 cycles<br>100000 32 cycles<br>100001–111111 Reserved |

[1] For CPO decodings other than 00000 and 11111, ‘READ_LAT’ is rounded up to the next integer value.

### 8.4.1.8 DDR SDRAM Control Configuration (DDR_SDRAM_CFG)

The DDR SDRAM control configuration register, shown in Figure 8-9, enables the interface logic and specifies certain operating features such as self refreshing, error checking and correcting, registered DIMMs, and dynamic power management.

Offset 0x110 Access: Read/Write

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | MEM_EN | SREN | ECC_EN | RD_EN | — | SDRAM_TYPE | | | — | | DYN_PWR | DBW | | 8_BE | NCAP | 3T_EN |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | 2T_EN | BA_INTLV_CTL | | | | | | | — | | x32_EN | PCHB8 | HSE | — | MEM_HALT | BI |
| Reset | All zeros | | | | | | | | | | | | | | | |

Figure 8-9. DDR SDRAM Control Configuration Register (DDR_SDRAM_CFG)

Table 8-13 describes the DDR_SDRAM_CFG fields.

Table 8-13. DDR_SDRAM_CFG Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0 | MEM_EN | DDR SDRAM interface logic enable.<br>0 SDRAM interface logic is disabled.<br>1 SDRAM interface logic is enabled. Must not be set until all other memory configuration parameters have been appropriately configured by initialization code. |
| 1 | SREN | Self refresh enable (during sleep).<br>0 SDRAM self refresh is disabled during sleep. Whenever self-refresh is disabled, the system is responsible for preserving the integrity of SDRAM during sleep.<br>1 SDRAM self refresh is enabled during sleep. |
| 2 | ECC_EN | ECC enable. Note that uncorrectable read errors may cause the assertion of *core_fault_in*, which causes the core to generate a machine check interrupt unless it is disabled (by clearing HID1[RFXE]). If RFXE is zero and this error occurs, ERR_DISABLE[MBED] and ERR_INT_EN[MBEE] must be zero and ECC_EN must be one to ensure an interrupt is generated. See Section 5.2, "e500 Core Integration and the Core Complex Bus (CCB)," and the *PowerPC™ e500 Core Family Reference Manual* for further details.<br>0 No ECC errors are reported. No ECC interrupts are generated.<br>1 ECC is enabled. |
| 3 | RD_EN | Registered DIMM enable. Specifies the type of DIMM used in the system.<br>0 Indicates unbuffered DIMMs.<br>1 Indicates registered DIMMs.<br>**Note:** RD_EN and 2T_EN must not both be set at the same time. |
| 4 | — | Reserved |
| 5–7 | SDRAM_TYPE | Type of SDRAM device to be used. This field is used when issuing the automatic hardware initialization sequence to DRAM through Mode Register Set and Extended Mode Register Set commands.<br>000–001Reserved<br>010 Reserved<br>011 DDR2 SDRAM<br>100 Reserved<br>101 Reserved<br>110 Reserved<br>111 DDR3 SDRAM |
| 8–9 | — | Reserved |
| 10 | DYN_PWR | Dynamic power management mode<br>0 Dynamic power management mode is disabled.<br>1 Dynamic power management mode is enabled. If there is no ongoing memory activity, the SDRAM CKE signal is negated. |
| 11–12 | DBW | DRAM data bus width.<br>00 64-bit bus is used<br>01 32-bit bus is used<br>10 Reserved<br>11 Reserved |
| 13 | 8_BE | 8-beat burst enable.<br>0 4-beat bursts are used on the DRAM interface.<br>1 8-beat bursts are used on the DRAM interface.<br>**Note:** DDR2 (SDRAM_TYPE = 011) must use 4-beat bursts, even when using 32-bit bus mode; DDR3 (SDRAM_TYPE = 111) must use 8-beat bursts when using 32-bit bus mode |

### Table 8-13. DDR_SDRAM_CFG Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 14 | NCAP | Non-concurrent auto-precharge. Some older DDR DRAMs do not support concurrent auto precharge. If one of these devices is used, then this bit needs to be set if auto precharge is used.<br>0 DRAMs in system support concurrent auto-precharge.<br>1 DRAMs in system do not support concurrent auto-precharge. |
| 15 | 3T_EN | Enable 3T timing. This field cannot be set if DDR_SDRAM_CFG[2T_EN] is also set. This field cannot be used with a 32-bit bus if 4-beat bursts are used.<br>0 1T timing is enabled if 2T_EN is cleared. The DRAM command/address are held for only 1 cycle on the DRAM bus.<br>1 3T timing is enabled. The DRAM command/address are held for 3 full cycles on the DRAM bus for every DRAM transaction. However, the chip select is only held for the third cycle. |
| 16 | 2T_EN | Enable 2T timing.This field should not be set if DDR_SDrAM_CFG[3T_EN] is set.<br>0 1T timing is enabled if 3T_EN is cleared. The DRAM command/address are held for only 1 cycle on the DRAM bus.<br>1 2T timing is enabled. The DRAM command/address are held for 2 full cycles on the DRAM bus for every DRAM transaction. However, the chip select is only held for the second cycle.<br>**Note:** RD_EN and 2T_EN must not both be set at the same time. |
| 17–23 | BA_INTLV_CTL | Bank (chip select) interleaving control. Set this field only if you wish to use bank interleaving. (‘*x*’ denotes a don’t care bit value. All unlisted field values are reserved.)<br>0000000 No external memory banks are interleaved<br>1000000 External memory banks 0 and 1 are interleaved<br>0100000 External memory banks 2 and 3 are interleaved<br>1100000 External memory banks 0 and 1 are interleaved together **and** banks 2 and 3 are interleaved together<br>*xx*00100 External memory banks 0 through 3 are all interleaved together |
| 24–25 | — | Reserved |
| 26 | x32_EN | x32 enable.<br>0 Either x8 or x16 discrete DRAM chips are used. In this mode, each data byte has a dedicated corresponding data strobe.<br>1 x32 discrete DRAM chips are used. In this mode, DQS0 is used to capture DQ[0:31], DQS4 is used to capture DQ[32:63] and DQS8 is used to capture ECC[0:7]. |
| 27 | PCHB8 | Precharge bit 8 enable.<br>0 MA[10] is used to indicate the auto-precharge and precharge all commands.<br>1 MA[8] is used to indicate the auto-precharge and precharge all commands.<br>If x32_EN is cleared, then PCHB8 should be cleared as well. |
| 28 | HSE | Global half-strength override<br>Sets I/O driver impedance to half strength. This impedance is used by the MDIC, address/command, data, and clock impedance values, but only if automatic hardware calibration is disabled and the corresponding group's software override is disabled in the DDR control driver register(s) described in Section 8.4.1.27, “DDR Control Driver Register 1 (DDRCDR_1).” This bit should be cleared if using automatic hardware calibration.<br>0 I/O driver impedance is configured to full strength.<br>1 I/O driver impedance is configured to half strength. |
| 29 | — | Reserved |

**Table 8-13. DDR_SDRAM_CFG Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 30 | MEM_HALT | DDR memory controller halt. When this bit is set, the memory controller does not accept any new data read/write transactions to DDR SDRAM until the bit is cleared again. This can be used when bypassing initialization and forcing MODE REGISTER SET commands through software.<br>0 DDR controller accepts new transactions.<br>1 DDR controller finishes any remaining transactions, and then it remains halted until this bit is cleared by software. |
| 31 | BI | Bypass initialization<br>0 DDR controller cycles through initialization routine based on SDRAM_TYPE<br>1 Initialization routine is bypassed. Software is responsible for initializing memory through DDR_SDRAM_MODE2 register. If software is initializing memory, then the MEM_HALT bit can be set to prevent the DDR controller from issuing transactions during the initialization sequence. Note that the DDR controller does not issue a DLL reset to the DRAMs when bypassing the initialization routine, regardless of the value of DDR_SDRAM_CFG[DLL_RST_DIS]. If a DLL reset is required, then the controller should be forced to enter and exit self refresh after the controller is enabled.<br>See Section 8.4.1.16, "DDR Initialization Address (DDR_INIT_ADDR)," for details on avoiding ECC errors in this mode. |

## 8.4.1.9 DDR SDRAM Control Configuration 2 (DDR_SDRAM_CFG_2)

The DDR SDRAM control configuration register 2, shown in Figure 8-10, provides more control configuration for the DDR controller.

Offset 0x114 Access: Read/Write

| | 0 | 1 | 2 | 3 | 4 5 | 6 8 | 9 10 | 11 15 |
|---|---|---|---|---|---|---|---|---|
| R<br>W | FRC_SR | SR_IE | DLL_RST_DIS | — | DQS_CFG | — | ODT_CFG | — |
| Reset | All zeros | | | | | | | |

| | 16 19 | 20 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|
| R<br>W | NUM_PR | — | OBC_CFG | AP_EN | D_INIT | — | RCW_EN | — | MD_EN |
| Reset | All zeros | | | | | | | | |

**Figure 8-10. DDR SDRAM Control Configuration Register 2 (DDR_SDRAM_CFG_2)**

Table 8-14 describes the DDR_SDRAM_CFG_2 fields.

**Table 8-14. DDR_SDRAM_CFG_2 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | FRC_SR | Force self refresh<br>0 DDR controller operates in normal mode.<br>1 DDR controller enters self-refresh mode. |
| 1 | SR_IE | Self-refresh interrupt enable. The DDR controller can be placed into self refresh mode by forcing the PIC to assert $\overline{\text{IRQ\_OUT}}$. This is considered a 'panic interrupt' by the DDR controller, and it enters self refresh as soon as possible. DDR_SDRAM_CFG[SREN] must also be set if the panic interrupt is used.<br>0 DDR controller does not enter self-refresh mode if panic interrupt is asserted.<br>1 DDR controller enters self-refresh mode if panic interrupt is asserted. |
| 2 | DLL_RST_DIS | DLL reset disable. The DDR controller typically issues a DLL reset to the DRAMs when exiting self refresh. However, this function may be disabled by setting this bit during initialization.<br>0 DDR controller issues a DLL reset to the DRAMs when exiting self refresh.<br>1 DDR controller does not issue a DLL reset to the DRAMs when exiting self refresh. |
| 3 | — | Reserved |
| 4–5 | DQS_CFG | DQS configuration<br>00 Reserved01Differential DQS signals are used for DDR2 support.<br>10 Reserved<br>11 Reserved |
| 6–8 | — | Reserved |
| 9–10 | ODT_CFG | ODT configuration. This field defines how ODT is driven to the on-chip IOs. See Section 8.4.1.27, "DDR Control Driver Register 1 (DDRCDR_1)," which defines the termination value that is used.)<br>00 Never assert ODT to internal IOs<br>01 Assert ODT to internal IOs only during writes to DRAM<br>10 Assert ODT to internal IOs only during reads to DRAM<br>11 Always keep ODT asserted to internal IOs |
| 11–15 | — | Reserved. |
| 16–19 | NUM_PR | Number of posted refreshes. This determines how many posted refreshes, if any, can be issued at one time. Note that if posted refreshes are used, then this field, along with DDR_SDRAM_INTERVAL[REFINT], must be programmed such that the maximum $t_{ras}$ specification cannot be violated.<br>0000 Reserved<br>0001 1 refresh is issued at a time<br>0010 2 refreshes is issued at a time<br>0011 3 refreshes is issued at a time<br>...<br>1000 8 refreshes is issued at a time<br>1001–1111 Reserved |
| 20–24 | — | Reserved |

**Table 8-14. DDR_SDRAM_CFG_2 Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 25 | OBC_CFG | On-The-Fly Burst Chop Configuration. Determines if on-the-fly Burst Chop is used. This bit should only be set if DDR3 memories are used. If on-the-fly Burst Chop mode is not used with DDR3 memories, then fixed Burst Chop mode may be used if the proper turnaround times are programmed into TIMING_CFG_0 and TIMING_CFG_4. DDR_SDRAM_CFG[8_BE] should be cleared for both on-the-fly Burst Chop mode or fixed Burst Chop mode when using a 64-bit data bus with DDR3 memories.<br>0 On-the-fly Burst Chop mode is disabled. Fixed burst lengths as defined in DDR_SDRAM_CFG[8_BE] are used. If fixed Burst Chop is used (with DDR3 memories), then DDR_SDRAM_CFG[8_BE] should be cleared.<br>1 On-the-fly Burst Chop mode is used. DDR_SDRAM_CFG[8_BE] should be cleared for on-the-fly Burst Chop mode. DDR_SDRAM_CFG[DBW] should also be cleared for on-the-fly Burst Chop mode |
| 26 | AP_EN | Address Parity Enable. Determines if address parity is generated and checked for the address and control signals when using registered DIMMs. If address parity is used, the MAPAR_OUT and MAPAR_ERR pins are used to drive the parity bit and to receive errors from the open-drain parity error signal. Even parity is used, and parity is generated for the MA[15:0], MBA[2:0], MRAS, MCAS, MWE signals. Parity does not generate for the MCKE[0:3], MODT[0:3], or MCS[0:3] signals. Note that address parity should not be used for non-zero values of TIMING_CFG_3[CNTL_ADJ].<br>0 Address parity is not used<br>1 Address parity is used |
| 27 | D_INIT | DRAM data initialization. This bit is set by software, and it is cleared by hardware. If software sets this bit before the memory controller is enabled, the controller automatically initializes DRAM after it is enabled. This bit is automatically cleared by hardware once the initialization is completed. This data initialization bit should only be set when the controller is idle.<br>0 There is not data initialization in progress, and no data initialization is scheduled<br>1 The memory controller initializes memory once it is enabled. This bit remains asserted until the initialization is complete. The value in DDR_DATA_INIT register is used to initialize memory. |
| 28 | — | Reserved |
| 29 | RCW_EN | Register Control Word Enable. If DDR3 registered DIMMs are used, it may be necessary to write the register control words before issuing commands to DRAM. If this bit is set, the controller will write the register control words after DDR_SDRAM_CFG[MEM_EN] is set, unless DDR_SDRAM_CFG[BI] is set. The register control words are written with the values in DDR_SDRAM_RCW_1 and DDR_SDRAM_RCW_2.<br>0 Register control words will not be automatically written during DRAM initialization<br>1 Register control words are automatically written during DRAM initialization. This bit should only be set if DDR3 registered DIMMs are used, and the default settings need to be modified. |
| 30 | — | Reserved |
| 31 | MD_EN | Mirrored DIMM Enable. Some DDR3 DIMMs are mirrored, where certain MA and MBA pins are mirrored on one side of the DIMM. When this bit is set, the controller will know to swap these signals before transmitting to the DRAM. The controller will assume that CS1 and CS3 are the 'mirrored' ranks of memory. The following signals are mirrored (MBA[0] vs MBA[1]; MA[3] vs MA[4]; MA[5] vs MA[6]; MA[7] vs MA[8]).<br>0 Mirrored DIMMs are not used<br>1 Mirrored DIMMs are used |

### 8.4.1.10 DDR SDRAM Mode Configuration (DDR_SDRAM_MODE)

The DDR SDRAM mode configuration register, shown in Figure 8-11, sets the values loaded into the DDR's mode registers.

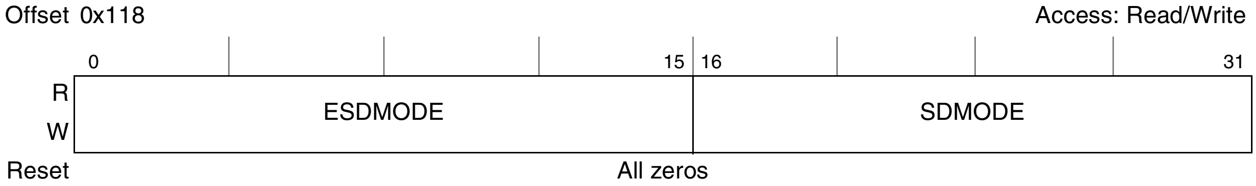


**Figure 8-11. DDR SDRAM Mode Configuration Register (DDR_SDRAM_MODE)**

Table 8-15 describes the DDR_SDRAM_MODE fields.

**Table 8-15. DDR_SDRAM_MODE Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–15 | ESDMODE | Extended SDRAM mode. Specifies the initial value loaded into the DDR SDRAM extended mode register. The range and meaning of legal values is specified by the DDR SDRAM manufacturer.<br>When this value is driven onto the address bus (during the DDR SDRAM initialization sequence), MA[0] presents the lsb of ESDMODE, which, in the big-endian convention shown in Figure 8-11, corresponds to ESDMODE[15]. The msb of the SDRAM extended mode register value must be stored at ESDMODE[0].<br>The value programmed into this field is also used for writing MR1 during write leveling for DDR3, although the bits specifically related to the write leveling scheme are handled automatically by the DDR controller. Even if DDR_SDRAM_CFG[BI] is set, this field is still used during write leveling. |
| 16–31 | SDMODE | SDRAM mode. Specifies the initial value loaded into the DDR SDRAM mode register. The range of legal values is specified by the DDR SDRAM manufacturer.<br>When this value is driven onto the address bus (during DDR SDRAM initialization), MA[0] presents the lsb of SDMODE, which, in the big-endian convention shown in Figure 8-11, corresponds to SDMODE[15]. The msb of the SDRAM mode register value must be stored at SDMODE[0]. Because the memory controller forces SDMODE[7] to certain values depending on the state of the initialization sequence, (for resetting the SDRAM's DLL) the corresponding bits of this field are ignored by the memory controller. Note that SDMODE[7] is mapped to MA[8]. |

### 8.4.1.11 DDR SDRAM Mode 2 Configuration (DDR_SDRAM_MODE_2)

The DDR SDRAM mode 2 configuration register, shown in Figure 8-12, sets the values loaded into the DDR's extended mode 2 and 3 registers (for DDR2).

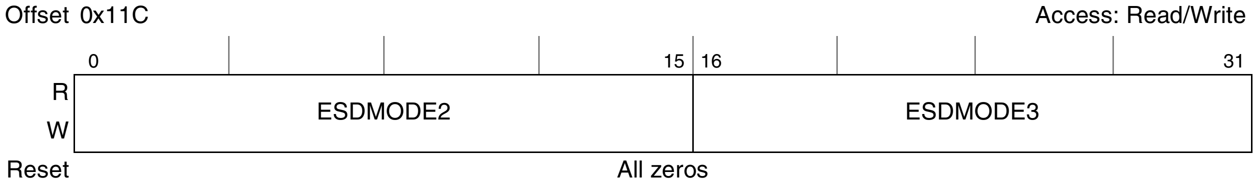


**Figure 8-12. DDR SDRAM Mode 2 Configuration Register (DDR_SDRAM_MODE_2)**

Table 8-16 describes the DDR_SDRAM_MODE_2 fields.

**Table 8-16. DDR_SDRAM_MODE_2 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–15 | ESDMODE2 | Extended SDRAM mode 2. Specifies the initial value loaded into the DDR SDRAM extended 2 mode register. The range and meaning of legal values is specified by the DDR SDRAM manufacturer. When this value is driven onto the address bus (during the DDR SDRAM initialization sequence), MA[0] presents the lsb bit of ESDMODE2, which, in the big-endian convention shown in Figure 8-12, corresponds to ESDMODE2[15]. The msb of the SDRAM extended mode 2 register value must be stored at ESDMODE2[0]. |
| 16–31 | ESDMODE3 | Extended SDRAM mode 3. Specifies the initial value loaded into the DDR SDRAM extended 3 mode register. The range of legal values of legal values is specified by the DDR SDRAM manufacturer. When this value is driven onto the address bus (during DDR SDRAM initialization), MA[0] presents the lsb of ESDMODE3, which, in the big-endian convention shown in Figure 8-12, corresponds to ESDMODE3[15]. The msb of the SDRAM extended mode 3 register value must be stored at ESDMODE3[0]. |

### 8.4.1.12 DDR SDRAM Mode Control Register (DDR_SDRAM_MD_CNTL)

The DDR SDRAM mode control register, shown in Figure 8-13, allows the user to carry out the following tasks:

- Issue a mode register set command to a particular chip select
- Issue an immediate refresh to a particular chip select
- Issue an immediate precharge or precharge all command to a particular chip select
- Force the CKE signals to a specific value

Table 8-17 describes the fields of this register. Table 8-18 shows the user how to set the fields of this register to accomplish the above tasks.

Offset 0x120 Access: Read/Write

|  | 0 | 1 2 3 | 4 | 5 7 | 8 | 9 | 10 11 | 12 | 13 15 | 16 31 |
|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | MD_EN | CS_SEL | — | MD_SEL | SET_REF | SET_PRE | CKE_CNTL | WRCW | — | MD_VALUE |
| Reset | All zeros |  |  |  |  |  |  |  |  |  |

**Figure 8-13. DDR SDRAM Mode Control Register (DDR_SDRAM_MD_CNTL)**

Table 8-17 describes the DDR_SDRAM_MD_CNTL fields.

**NOTE**

Note that MD_EN, SET_REF, and SET_PRE are mutually exclusive; only one of these fields can be set at a time.

### Table 8-17. DDR_SDRAM_MD_CNTL Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0 | MD_EN | Mode enable. Setting this bit specifies that valid data in MD_VALUE is ready to be written to DRAM as one of the following commands:<br>• MODE REGISTER SET<br>• EXTENDED MODE REGISTER SET<br>• EXTENDED MODE REGISTER SET 2<br>• EXTENDED MODE REGISTER SET 3<br>The specific command to be executed is selected by setting MD_SEL. In addition, the chip select must be chosen by setting CS_SEL. MD_EN is set by software and cleared by hardware once the command has been issued.<br>0 Indicates that no mode register set command needs to be issued.<br>1 Indicates that valid data contained in the register is ready to be issued as a mode register set command. |
| 1 | — | Reserved |
| 2–3 | CS_SEL | Select chip select. Specifies the chip select that is driven active due to any command forced by software in DDR_SDRAM_MD_CNTL.<br>00 Chip select 0 is active<br>01 Chip select 1 is active<br>10 Chip select 2 is active<br>11 Chip select 3 is active |
| 1–3 | CS_SEL | Select chip select. Specifies the chip select that is driven active due to any command forced by software in DDR_SDRAM_MD_CNTL.<br>000 Chip select 0 is active<br>001 Chip select 1 is active<br>010 Chip select 2 is active<br>011 Chip select 3 is active<br>100 Chip select 0 and chip select 1 are active<br>101 Chip select 2and chip select 3 are active<br>110-111Chip select 3 is active |
| 4 | — | Reserved |
| 5–7 | MD_SEL | Mode register select. MD_SEL specifies one of the following:<br>• During a mode select command, selects the SDRAM mode register to be changed<br>• During a precharge command, selects the SDRAM logical bank to be precharged. A precharge all command ignores this field.<br>• During a refresh command, this field is ignored.<br>Note that MD_SEL contains the value that is presented onto the memory bank address pins (MBA*n*) of the DDR controller.<br>000 MR<br>001 EMR<br>010 EMR2<br>011 EMR3 |
| 8 | SET_REF | Set refresh. Forces an immediate refresh to be issued to the chip select specified by DDR_SDRAM_MD_CNTL[CS_SEL]. This bit is set by software and cleared by hardware once the command has been issued.<br>0 Indicates that no refresh command needs to be issued.<br>1 Indicates that a refresh command is ready to be issued. |
| 9 | SET_PRE | Set precharge. Forces a precharge or precharge all to be issued to the chip select specified by DDR_SDRAM_MD_CNTL[CS_SEL]. This bit is set by software and cleared by hardware once the command has been issued.<br>0 Indicates that no precharge all command needs to be issued.<br>1 Indicates that a precharge all command is ready to be issued. |

Table 8-17. DDR_SDRAM_MD_CNTL Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 10–11 | CKE_CNTL | Clock enable control. Allows software to globally clear or set all CKE signals issued to DRAM. Once software has forced the value driven on CKE, that value continues to be forced until software clears the CKE_CNTL bits. At that time, the DDR controller continues to drive the CKE signals to the same value forced by software until another event causes the CKE signals to change (such as, self refresh entry/exit, power down entry/exit).<br>00 CKE signals are not forced by software.<br>01 CKE signals are forced to a low value by software.<br>10 CKE signals are forced to a high value by software.<br>11 Reserved |
| 12 | WRCW | Write register control word. If software sets this bit, then a register control word is written by asserting the selected chip selects while providing the programmed data on the MA and MBA signals. The RAS, CAS, and WE will remain deasserted during this write The MD_EN field should also be set to force a register control word write. This should only be set if DDR3 registered DIMM s are used, and the register needs to be configured. If DDR_SDRAM_MD_CNTL is used to write RCW2 specifically, then software must guarantee that the timing parameter, *t-STAB*, is met before future accesses to the controller are allowed. In addition, DDR_SDRAM_MD_CNTL register cannot be used to write the RCWs if write leveling is used, since write leveling is run automatically before DDR_SDRAM_MD_CNTL can be used to force RCW writes..<br>0 Indicates that a register control word write will not be issued if MD_EN is set.<br>1 Indicates that a register control word write is issued if MD_EN is set. |
| 13–15 | — | Reserved |
| 16–31 | MD_VALUE | Mode register value. This field, which specifies the value that is presented on the memory address pins of the DDR controller during a mode register set command, is significant only when this register is used to issue a mode register set command or a precharge or precharge all command.<br>For a mode register set command, this field contains the data to be written to the selected mode register.<br>For a precharge command, only bit five is significant:<br>0 Issue a precharge command; MD_SEL selects the logical bank to be precharged<br>1 Issue a precharge all command; all logical banks are precharged |

Table 8-18 shows how DDR_SDRAM_MD_CNTL fields should be set for each of the tasks described above.

Table 8-18. Settings of DDR_SDRAM_MD_CNTL Fields

| Field | Mode Register Set | Refresh | Precharge | Clock Enable Signals Control |
|---|---|---|---|---|
| MD_EN | 1 | 0 | 0 | — |
| SET_REF | 0 | 1 | 0 | — |
| SET_PRE | 0 | 0 | 1 | — |
| CS_SEL | Chooses chip select (CS) | | | — |
| MD_SEL | Select mode register. See Table 8-17. | — | Selects logical bank | — |
| MD_VALUE | Value written to mode register | — | Only bit five is significant. See Table 8-17. | — |
| CKE_CNTL | 0 | 0 | 0 | See Table 8-17. |

### 8.4.1.13 DDR SDRAM Interval Configuration (DDR_SDRAM_INTERVAL)

The DDR SDRAM interval configuration register, shown in Figure 8-14, sets the number of DRAM clock cycles between bank refreshes issued to the DDR SDRAMs. In addition, the number of DRAM cycles that a page is maintained after it is accessed is provided here.

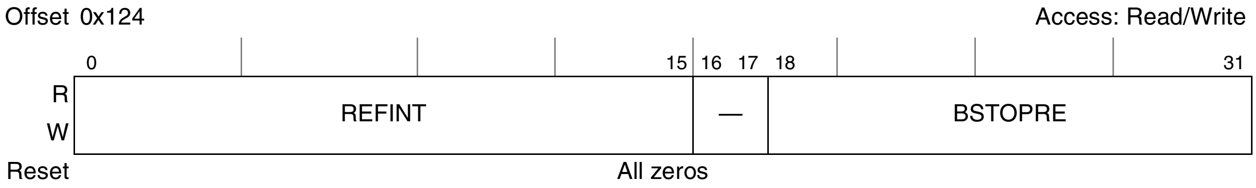


Figure 8-14. DDR SDRAM Interval Configuration Register (DDR_SDRAM_INTERVAL)

Table 8-19 describes the DDR_SDRAM_INTERVAL fields.

Table 8-19. DDR_SDRAM_INTERVAL Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–15 | REFINT | Refresh interval. Represents the number of memory bus clock cycles between refresh cycles. Depending on DDR_SDRAM_CFG_2[NUM_PR], some number of rows are refreshed in each DDR SDRAM physical bank during each refresh cycle. The value for REFINT depends on the specific SDRAMs used and the interface clock frequency. Refreshes are not issued when the REFINT is set to all 0s. |
| 16–17 | — | Reserved |
| 18–31 | BSTOPRE | Precharge interval. Sets the duration (in memory bus clocks) that a page is retained after a DDR SDRAM access. If BSTOPRE is zero, the DDR memory controller uses auto-precharge read and write commands rather than operating in page mode. This is called global auto-precharge mode. |

### 8.4.1.14 DDR SDRAM Data Initialization (DDR_DATA_INIT)

The DDR SDRAM data initialization register, shown in Figure 8-15, provides the value that is used to initialize memory if DDR_SDRAM_CFG2[D_INIT] is set.

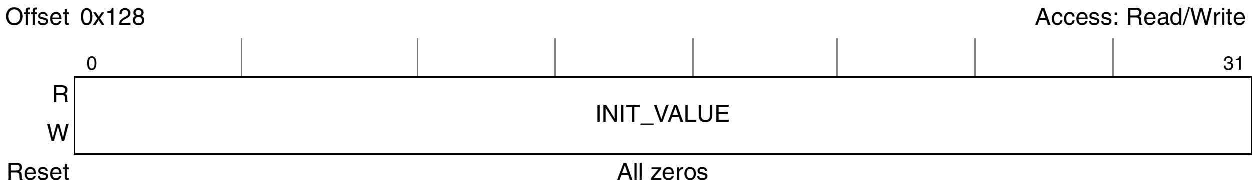


Figure 8-15. DDR SDRAM Data Initialization Configuration Register (DDR_DATA_INIT)

Table 8-20 describes the DDR_DATA_INIT fields.

Table 8-20. DDR_DATA_INIT Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–31 | INIT_VALUE | Initialization value. Represents the value that DRAM is initialized with if DDR_SDRAM_CFG2[D_INIT] is set. |

### 8.4.1.15 DDR SDRAM Clock Control (DDR_SDRAM_CLK_CNTL)

The DDR SDRAM clock control configuration register, shown in Figure 8-16, provides a 1/8-cycle clock adjustment.

Offset 0x130 Access: Read/Write

| | 0–4 | 5–8 | 9–31 |
|---|---|---|---|
| R | — | CLK_ADJUST | — |
| W | | | |
| Reset | 0 0 0 0 0 | 0 1 0 0 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 |

**Figure 8-16. DDR SDRAM Clock Control Configuration Register (DDR_SDRAM_CLK_CNTL)**

Table 8-21 describes the DDR_SDRAM_CLK_CNTL fields.

**Table 8-21. DDR_SDRAM_CLK_CNTL Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–4 | — | Reserved |
| 5–8 | CLK_ADJUST | Clock adjust<br>0000 Clock is launched aligned with address/command<br>0001 Clock is launched 1/8 applied cycle after address/command<br>0010 Clock is launched 1/4 applied cycle after address/command<br>0011 Clock is launched 3/8 applied cycle after address/command<br>0100 Clock is launched 1/2 applied cycle after address/command<br>0101 Clock is launched 5/8 applied cycle after address/command<br>0110 Clock is launched 3/4 applied cycle after address/command<br>0111 Clock is launched 7/8 applied cycle after address/command<br>1000 Clock is launched 1 applied cycle after address/command<br>1001–1111 Reserved |
| 9–31 | — | Reserved |

### 8.4.1.16 DDR Initialization Address (DDR_INIT_ADDR)

The DDR SDRAM initialization address register, shown in Figure 8-17, provides the address that is used for the data strobe to data skew adjustment and automatic $\overline{\text{CAS}}$ to preamble calibration after POR.

**NOTE**

After the skew adjustment, this address contains bad ECC data. This is not important at POR, as all of memory should be subsequently initialized if ECC is enabled (either by software or through the use of DDR_SDRAM_CFG_2[D_INIT]).

If an $\overline{\text{HRESET}}$ has been issued after the DRAM is in self-refresh mode, however, memory is not initialized, so this address should be written to using an 8- or 32-byte transaction to avoid possible ECC errors if this address could later be accessed.

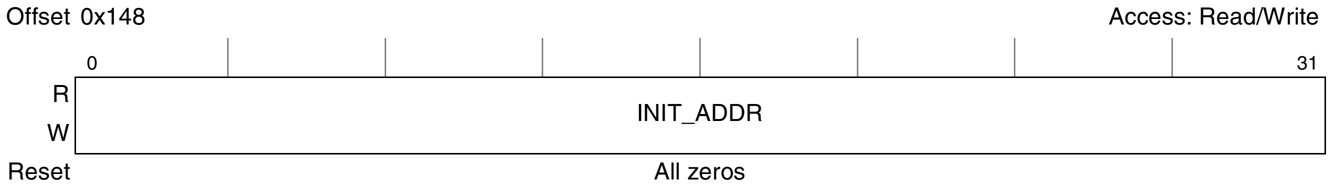


**Figure 8-17. DDR Initialization Address Configuration Register (DDR_INIT_ADDR)**

Table 8-22 describes the DDR_INIT_ADDR fields.

**Table 8-22. DDR_INIT_ADDR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–31 | INIT_ADDR | Initialization address. Represents the address that is used for the data strobe to data skew adjustment and automatic CAS to preamble calibration at POR. This address is written to during the initialization sequence. |

### 8.4.1.17 DDR Initialization Enable Extended Address (DDR_INIT_EXT_ADDR)

The DDR SDRAM initialization extended address register, shown in Figure 8-18, provides the extended address that is used for the data strobe to data skew adjustment and automatic $\overline{\text{CAS}}$ to preamble calibration after POR.

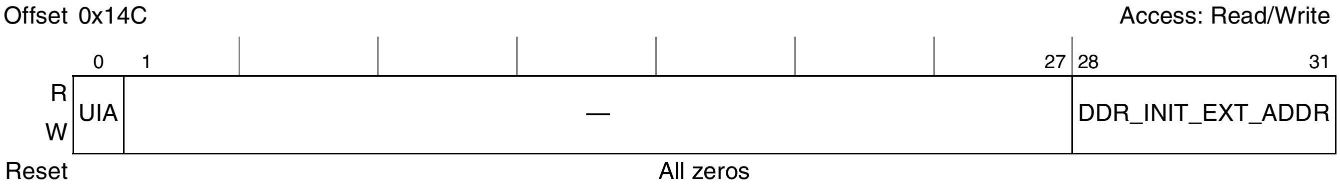


**Figure 8-18. DDR Initialization Extended Address Configuration Register (DDR_INIT_EXT_ADDR)**

Table 8-23 describes the DDR_INIT_EXT_ADDR fields.

**Table 8-23. DDR_INIT_EXT_ADDR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | UIA | Use initialization address.<br>0 Use the default address for training sequence as calculated by the controller. This is the first valid address in the first enabled chip select.<br>1 Use the initialization address programmed in DDR_INIT_ADDR and DDR_INIT_EXT_ADDR. |
| 1–27 | — | Reserved, should be cleared. |
| 28–31 | INIT_EXT_ADDR | Initialization extended address. Represents the extended address that is used for the data strobe to data skew adjustment and automatic $\overline{\text{CAS}}$ to preamble calibration at POR. This extended address is written to during the initialization sequence. |

### 8.4.1.18 DDR SDRAM Timing Configuration 4 (TIMING_CFG_4)

The DDR SDRAM timing configuration 4 register, shown in Figure 8-19, provides additional timing fields required to support DDR3 memories.

Offset 0x160 Access: Read/Write

| | 0–3 | 4–7 | 8–11 | 12–15 | 16–29 | 30–31 |
|---|---|---|---|---|---|---|
| R<br>W | RWT | WRT | RRT | WWT | — | DLL_LOCK |
| Reset | All zeros | | | | | |

**Figure 8-19. DDR SDRAM Timing Configuration 4 Register (TIMING_CFG_4)**

Table 8-24 describes the TIMING_CFG_4 fields.

**Table 8-24. TIMING_CFG_4 Field Descriptions**

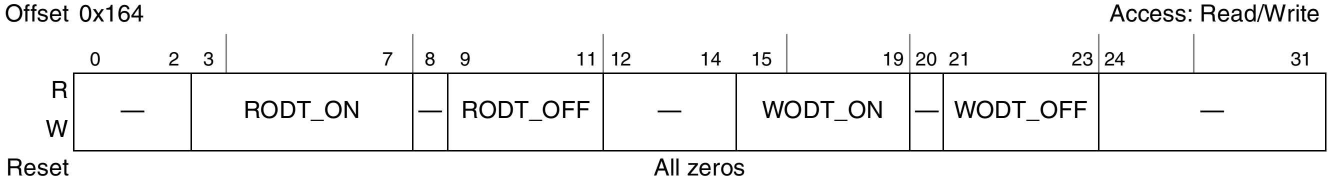
| Bits | Name | Description |
|---|---|---|
| 0–3 | RWT | Read-to-write turnaround for same chip select. Specifies how many cycles are added between a read to write turnaround for transactions to the same chip select. If a value of 0000 is chosen, then the DDR controller uses the value used for transactions to different chip selects, as defined in TIMING_CFG_0[RWT]. This field can be used to improve performance when operating in burst-chop mode by forcing transactions to the same chip select to use extra cycles, while transaction to different chip selects can utilize the tri-state time on the DRAM interface. Regardless of the value that is set in this field, the value defined by TIMING_CFG_0[RWT] also is met before issuing a write command.<br>0000 Default<br>0001 1 clock<br>0010 2 clocks<br>0011 3 clocks<br>0100 4 clocks<br>0101 5 clocks<br>0110 6 clocks<br>0111 7 clocks<br>1000 8 clocks<br>1001 9 clocks<br>1010 10 clocks<br>1011 11 clocks<br>1100 12 clocks<br>1101 13 clocks<br>1110 14 clocks<br>1111 15 clocks |
| 4–7 | WRT | Write-to-read turnaround for same chip select. Specifies how many cycles are added between a write to read turnaround for transactions to the same chip select. If a value of 0000 is chosen, then the DDR controller uses the value used for transactions to different chip selects, as defined in TIMING_CFG_0[WRT]. This field can be used to improve performance when operating in burst-chop mode by forcing transactions to the same chip select to use extra cycles, while transaction to different chip selects can utilize the tri-state time on the DRAM interface. Regardless of the value that is set in this field, the value defined by TIMING_CFG_0[WRT] also is met before issuing a read command.<br>0000 Default<br>0001 1 clock<br>0010 2 clocks<br>0011 3 clocks<br>0100 4 clocks<br>0101 5 clocks<br>0110 6 clocks<br>0111 7 clocks<br>1000 8 clocks<br>1001 9 clocks<br>1010 10 clocks<br>1011 11 clocks<br>1100 12 clocks<br>1101 13 clocks<br>1110 14 clocks<br>1111 15 clocks |

Table 8-24. TIMING_CFG_4 Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 8–11 | RRT | Read-to-read turnaround for same chip select. Specifies how many cycles are added between reads to the same chip select. If a value of 0000 is chosen, then 2 cycles are required between read commands to the same chip select if 4-beat bursts are used (4 cycles are required if 8-beat bursts are used). Note that DDR3 does not support 4-beat bursts. However, this field may be used to add extra cycles when burst-chop mode is used, and the DDR controller must wait 4 cycles for read-to-read transactions to the same chip select.<br>0000 BL/2 clocks 1000 BL/2 + 8 clocks<br>0001 BL/2 + 1 clock 1001 BL/2 + 9 clocks<br>0010 BL/2 + 2 clocks 1010 BL/2 + 10 clocks<br>0011 BL/2 + 3 clocks 1011 BL/2 + 11 clocks<br>0100 BL/2 + 4 clocks 1100 BL/2 + 12 clocks<br>0101 BL/2 + 5 clocks 1101 BL/2 + 13 clocks<br>0110 BL/2 + 6 clocks 1110 BL/2 + 14 clocks<br>0111 BL/2 + 7 clocks 1111 BL/2 + 15 clocks |
| 12–15 | WWT | Write-to-write turnaround for same chip select. Specifies how many cycles are added between writes to the same chip select. If a value of 0000 is chosen, then 2 cycles are required between write commands to the same chip select if 4-beat bursts are used (4 cycles are required if 8-beat bursts are used). Note that DDR3 does not support 4-beat bursts. However, this field may be used to add extra cycles when burst-chop mode is used, and the DDR controller must wait 4 cycles for write-to-write transactions to the same chip select.<br>0000 BL/2 clocks 1000 BL/2 + 8 clocks<br>0001 BL/2 + 1 clock 1001 BL/2 + 9 clocks<br>0010 BL/2 + 2 clocks 1010 BL/2 + 10 clocks<br>0011 BL/2 + 3 clocks 1011 BL/2 + 11 clocks<br>0100 BL/2 + 4 clocks 1100 BL/2 + 12 clocks<br>0101 BL/2 + 5 clocks 1101 BL/2 + 13 clocks<br>0110 BL/2 + 6 clocks 1110 BL/2 + 14 clocks<br>0111 BL/2 + 7 clocks 1111 BL/2 + 15 clocks |
| 16–29 | — | Reserved, should be cleared. |
| 30–31 | DLL_LOCK | DDR SDRAM DLL Lock Time. This provides the number of cycles that it takes for the DRAMs DLL to lock at POR and after exiting self refresh. The controller waits the specified number of cycles before issuing any commands after exiting POR or self refresh.<br>00 200 clocks 10 Reserved<br>01 512 clocks 11 Reserved |

### 8.4.1.19 DDR SDRAM Timing Configuration 5 (TIMING_CFG_5)

The DDR SDRAM timing configuration 5 register, shown in Figure 8-20, provides additional timing fields required to support DDR3 memories.



Offset 0x164 Access: Read/Write

| | 0 2 | 3 7 | 8 | 9 11 | 12 14 | 15 19 | 20 | 21 23 | 24 31 |
|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | RODT_ON | — | RODT_OFF | — | WODT_ON | — | WODT_OFF | — |
| Reset | All zeros | | | | | | | | |

Figure 8-20. DDR SDRAM Timing Configuration 5 Register (TIMING_CFG_5)

Table 8-25 describes the TIMING_CFG_5 fields.

**Table 8-25. TIMING_CFG_5 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–2 | — | Reserved, should be cleared. |
| 3–7 | RODT_ON | Read to ODT on. Specifies the number of cycles that passes from when a read command is placed on the DRAM bus until the assertion of the relevant ODT signal(s). The default case (00000) provides a decode of RL - 3 cycles to support legacy of past products. RL is the read latency, derived from CAS latency + additive latency. If 2T timing is used, an extra cycle is automatically added to the value selected in this field.<br>00000 RL - 3 clocks<br>00001 0 clocks<br>00010 1 clocks<br>00011 2 clocks<br>.<br>.<br>.<br>011111 14 clocks<br>10000 15 clocks<br>10001 16 clocks<br>10010 17 clocks<br>10011 18 clocks<br>.<br>.<br>.<br>11111 30 clocks |
| 8 | — | Reserved, should be cleared. |
| 9–11 | RODT_OFF | Read to ODT off. Specifies the number of cycles that the relevant ODT signal(s) remains asserted for each read transaction. The default case (000) leaves the ODT signal(s) asserted for 3 DRAM cycles.<br>000 3 clocks<br>001 1 clock<br>010 2 clocks<br>011 3 clocks<br>100 4 clocks<br>101 5 clocks<br>110 6 clocks<br>111 7 clocks |
| 12–14 | — | Reserved, should be cleared. |
| 15–19 | WODT_ON | Write to ODT On Specifies the number of cycles that passes from when a write command is placed on the DRAM bus until the assertion of the relevant ODT signal(s). The default case (00000) provides a decode of WL - 3 cycles to support legacy of past products. WL is the write latency, derived from Write Latency + Additive Latency. If 2T timing is used, an extra cycle is automatically added to the value selected in this field.<br>00000 WL – 3 clocks<br>00001 0 clocks<br>00010 1 clocks<br>00011 2 clocks<br>.<br>.<br>.<br>01111 14 clocks<br>10000 15 clocks<br>10001 16 clocks<br>10010 17 clocks<br>10011 18 clocks<br>.<br>.<br>.<br>11111 30 clocks |
| 20 | — | Reserved, should be cleared. |
| 21–23 | WODT_OFF | Write to ODT Off. Specifies the number of cycles that the relevant ODT signal(s) remains asserted for each write transaction. The default case (000) leaves the ODT signal(s) asserted for 3 DRAM cycles.<br>000 3 clocks<br>001 1 clock<br>010 2 clocks<br>011 3 clocks<br>100 4 clocks<br>101 5 clocks<br>110 6 clocks<br>111 7 clocks |
| 24–31 | — | Reserved, should be cleared. |

### 8.4.1.20 DDR ZQ Calibration Control (DDR_ZQ_CNTL)

The DDR ZQ Calibration Control register, shown in Figure 8-21, provides the enable and controls required for ZQ calibration when using DDR3 SDRAM devices.

There is a limitation for various DRAM timing parameters when ZQ calibration is used. The factors involved in this limitation are DDR_ZQ_CNTL[ZQOPER], DDR_ZQ_CNTL[ZQCS], TIMING_CFG_1[PRETOACT], TIMING_CFG_1[REFREC], DDR_SDRAM_INTERVAL[REFINT], and the number of chip selects enabled. If the following condition is true:

[((DDR_ZQ_CNTL[ZQOPER] + DDR_ZQ_CNTL[ZQCS])* (# enabled chip selects)) +
TIMING_CFG_1[PRETOACT] +
TIMING_CFG_1[REFREC] + $2t_{CK}$] > (DDR_SDRAM_INTERVAL[REFINT]),

then it is possible that one refresh is skipped when the controller is exiting self refresh. If this is an issue, then posted refreshes could be used to extend the refresh interval. Another alternative is to use the DDR_SDRAM_MD_CNTL register to force an extra refresh to each chip select after exiting self refresh mode. However, DDR3 timing parameters for most devices/frequencies do not allow for a refresh to be missed.

Offset 0x170 Access: Read/Write

| | 0 | 1 3 | 4 7 | 8 11 | 12 15 | 16 19 | 20 23 | 24 31 |
|---|---|---|---|---|---|---|---|---|
| R<br>W | ZQ_EN | — | ZQINIT | — | ZQOPER | — | ZQCS | — |
| Reset | All zeros | | | | | | | |

**Figure 8-21. DDR ZQ Calibration Control Register (DDR_ZQ_CNTL)**

Table 8-26 describes the DDR_ZQ_CNTL fields.

**Table 8-26. DDR_ZQ_CNTL Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | ZQ_EN | ZQ Calibration Enable. This bit determines if ZQ calibration is used. This bit should only be set if DDR3 memory is used (DDR_SDRAM_CFG[SDRAM_TYPE] = 3'b111).<br>0 ZQ Calibration is not used.<br>1 ZQ Calibration is used. A ZQCL command is issued by the DDR controller after POR and anytime the DDR controller is exiting self refresh. A ZQCS command is issued every 32 refresh sequences to account for VT variations. |
| 1–3 | — | Reserved, should be cleared. |
| 4–7 | ZQINIT | POR ZQ Calibration Time ($t_{ZQinit}$). Determines the number of cycles that must be allowed for DRAM ZQ calibration at POR. Each chip select is calibrated separately, and this time must elapse after the ZQCL command is issued for each chip select before a separate command may be issued.<br>0000–0110 Reserved<br>0111 128 clocks<br>1000 256 clocks<br>1001 512 clocks<br>1010 1024 clocks<br>1011–1111 Reserved |
| 8–11 | — | Reserved, should be cleared. |

Table 8-26. DDR_ZQ_CNTL Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 12–15 | ZQOPER | Normal Operation Full Calibration Time ($t_{ZQoper}$). Determines the number of cycles that must be allowed for DRAM ZQ calibration when exiting self refresh. Each chip select is calibrated separately, and this time must elapse after the ZQCL command is issued for each chip select before a separate command may be issued.<br>0000-0110 Reserved<br>0111 128 clocks<br>1000 256 clocks<br>1001 512 clocks<br>1010 1024 clocks<br>1011-1111 Reserved |
| 16–19 | — | Reserved, should be cleared. |
| 20–23 | ZQCS | Normal Operation Short Calibration Time ($t_{ZQCS}$). Determines the number of cycles that must be allowed for DRAM ZQ calibration during dynamic calibration which is issued every 32 refresh cycles. Each chip select is calibrated separately, and this time must elapse after the ZQCS command is issued for each chip select before a separate command may be issued.<br>0000 1 clocks<br>0001 2 clocks<br>0010 4 clocks<br>0011 8 clocks<br>0100 16 clocks<br>0101 32 clocks<br>0110 64 clocks<br>0111 128 clocks<br>1000 256 clocks<br>1001 512 clocks<br>1010-1111 Reserved |
| 24–31 | — | Reserved, should be cleared. |

## 8.4.1.21 DDR Write Leveling Control (DDR_WRLVL_CNTL)

The DDR Write Leveling Control register, shown in Figure 8-22, provides controls for write leveling, as it is supported for DDR3 memory devices.

Offset 0x174 Access: Read/Write

| | 0 | 1 4 | 5 7 | 8 | 9 11 | 12 | 13 15 | 16 19 | 20 | 21 23 | 24 26 27 | 28 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | WRLVL_EN | — | WRLVL_MRD | — | WRLVL_ODTEN | — | WRLVL_DQSEN | WRLVL_SMPL | — | WRLVL_WLR | — | WRLVL_START |
| Reset | All zeros | | | | | | | | | | | |

Figure 8-22. DDR Write Leveling Control Register (DDR_WRLVL_CNTL)

Table 8-27 describes the DDR_WRLVL_CNTL fields.

**Table 8-27. DDR_WRLVL_CNTL Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | WRLVL_EN | Write Leveling Enable. This bit determines if write leveling is used. If this bit is set, then the DDR controller performs write leveling immediately after initializing the DRAM. This bit should only be set if DDR3 memory is used (DDR_SDRAM_CFG[SDRAM_TYPE] = 3’b111).<br>0 Write leveling is not used<br>1 Write leveling is used |
| 1–4 | — | Reserved, should be cleared. |
| 5–7 | WRLVL_MRD | First DQS pulse rising edge after margining mode is programmed ($t_{WL\_MRD}$). Determines how many cycles to wait after margining mode has been programmed before the first DQS pulse may be issued. This field is only relevant when DDR_WRLVL_CNTL[WRLVL_EN] is set.<br>000 1 clocks<br>001 2 clocks<br>010 4 clocks<br>011 8 clocks<br>100 16 clocks<br>101 32 clocks<br>110 64 clocks<br>111 128 clocks |
| 8 | — | Reserved, should be cleared. |
| 9–11 | WRLVL_ODTEN | ODT delay after margining mode is programmed ($t_{WL\_ODTEN}$). Determines how many cycles to wait after margining mode has been programmed until ODT may be asserted.This field is only relevant when DDR_WRLVL_CNTL[WRLVL_EN] is set.<br>000 1 clocks<br>001 2 clocks<br>010 4 clocks<br>011 8 clocks<br>100 16 clocks<br>101 32 clocks<br>110 64 clocks<br>111 128 clocks |
| 12 | — | Reserved, should be cleared. |
| 13–15 | WRLVL_DQSEN | DQS/$\overline{\text{DQS}}$ delay after margining mode is programmed ($t_{WL\_DQSEN}$). Determines how many cycles to wait after margining mode has been programmed until DQS may be actively driven. This field is only relevant when DDR_WRLVL_CNTL[WRLVL_EN] is set.<br>000 1 clocks<br>001 2 clocks<br>010 4 clocks<br>011 8 clocks<br>100 16 clocks<br>101 32 clocks<br>110 64 clocks<br>111 128 clocks |

### Table 8-27. DDR_WRLVL_CNTL Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 16–19 | WRLVL_SMPL | Write leveling sample time. Determines the number of cycles that must pass before the data signals are sampled after a DQS pulse during margining mode. This field should be programmed at least 6 cycles higher than $t_{WLO}$ to allow enough time for propagation delay and sampling of the prime data bits. This field is only relevant when DDR_WRLVL_CNTL[WRLVL_EN] is set.<br>0000 Reserved (if DDR_WRLVL_CNTL[WRLVL_EN] is set)<br>0001 1 clocks<br>0010 2 clocks<br>0011 3 clocks<br>0100 4 clocks<br>0101 5 clocks<br>1010 6 clocks<br>0111 7 clocks<br>1000 8 clocks<br>1001 9 clocks<br>1010 10 clocks<br>1011 11 clocks<br>1100 12 clocks<br>1101 13 clocks<br>1010 14 clocks<br>1111 15 clocks |
| 20 | — | Reserved, should be cleared. |
| 21–23 | WRLVL_WLR | Write leveling repetition time. Determines the number of cycles that must pass between DQS pulses during write leveling. This field is only relevant when DDR_WRLVL_CNTL[WRLVL_EN] is set.<br>000 1 clocks<br>001 2 clocks<br>010 4 clocks<br>011 8 clocks<br>100 16 clocks<br>101 32 clocks<br>110 64 clocks<br>111 128 clocks |

**Table 8-27. DDR_WRLVL_CNTL Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 24–27 | — | Reserved, should be cleared. |
| 28–31 | WRLVL_START | Write leveling start time. Determines the value to use for the DQS_ADJUST for the first sample when write leveling is enabled.<br>00000 0 clock delay<br>00001 1/8 clock delay<br>00010 1/4 clock delay<br>00011 3/8 clock delay<br>00100 1/2 clock delay<br>00101 5/8 clock delay<br>00110 3/4 clock delay<br>00111 7/8 clock delay<br>01000 1 clock delay<br>01001 9/8 clock delay<br>01010 5/4 clock delay<br>01011 11/8 clock delay<br>01100 3/2 clock delay<br>01101 13/8 clock delay<br>01110 7/4 clock delay<br>01111 15/8 clock delay<br>10000 2 clock delay<br>10001 17/8 clock delay<br>10010 9/4 clock delay<br>10011 19/8 clock delay<br>10100 5/2 clock delay<br>10101-11111Reserved |

### 8.4.1.22 DDR Self Refresh Counter (DDR_SR_CNTR)

The DDR Self Refresh Counter register can be programmed to force the DDR controller to enter self refresh after a predefined period of idle time.

Offset 0x17C Access: Read/Write

| | 0 – 11 | 12 – 15 | 16 – 31 |
|---|---|---|---|
| R<br>W | — | SR_IT | — |
| Reset | All zeros | | |

**Figure 8-23. DDR Self Refresh Counter Register (DDR_SR_CNTR)**

Table 8-28 describes the DDR_SR_CNTR fields.

**Table 8-28. DDR_SR_CNTR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–11 | — | Reserved, should be cleared. |
| 12–15 | SR_IT | Self Refresh Idle Threshold. Defines the number of DRAM cycles that must pass while the DDR controller is idle before it will enter self refresh. Anytime a transaction is issued to the DDR controller, it will reset its internal counter. When a new transaction is received by the DDR controller, it will exit self refresh and reset its internal counter. If this field is zero, then the described power savings feature is disabled. In addition, if a non-zero value is programmed into this field, then the DDR controller will exit self refresh anytime a transaction is issued to the DDR controller, regardless of the reason self refresh was initially entered.<br>If this field is set to a non-zero value, then DDR_SDRAM_CFG[SREN] must also be set.<br>0000 Automatic self refresh entry disabled<br>0001 2^10 DRAM clocks<br>0010 2^12 DRAM clocks<br>0011 2^14 DRAM clocks<br>0100 2^16 DRAM clocks<br>0101 2^18 DRAM clocks<br>0110 2^20 DRAM clocks<br>0111 2^22 DRAM clocks<br>1000 2^24 DRAM clocks<br>1001 2^26 DRAM clocks<br>1010 2^28 DRAM clocks<br>1011 2^30 DRAM clocks<br>1100-1111 Reserved |
| 16–31 | — | Reserved, should be cleared. |

### 8.4.1.23 DDR SDRAM Register Control Word 1 (DDR_SDRAM_RCW_1)

The DDR Register Control Word 1 register should be programmed with the intended values of the register control words if DDR_SDRAM_CFG[RCW_EN] is set. Each 4-bit field represents the value that is placed on MA[3], MA[4], MBA[0], and MBA[1] during register control word writes.

Offset 0x180 Access: Read/Write

| | 0–3 | 4–7 | 8–11 | 12–15 | 16–19 | 20–23 | 24–27 | 28–31 |
|---|---|---|---|---|---|---|---|---|
| R<br>W | RCW0 | RCW1 | RCW2 | RCW3 | RCW4 | RCW5 | RCW6 | RCW7 |
| Reset | All zeros | | | | | | | |

**Figure 8-24. DDR Register Control Word 1 (DDR_SDRAM_RCW_1)**

Table 8-28 describes the DDR_SDRAM_RCW_1 fields.

Table 8-29. DDR_Register Control Word 1 Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–3 | RCW0 | Register Control Word 0. Represents the value that is placed on MBA[1], MBA[0], MA[4], and MA[3] during writes to register control word 0. |
| 4–7 | RCW1 | Register Control Word 0. Represents the value that is placed on MBA[1], MBA[0], MA[4], and MA[3] during writes to register control word 1. |
| 8–11 | RCW2 | Register Control Word 0. Represents the value that is placed on MBA[1], MBA[0], MA[4], and MA[3] during writes to register control word 2. |
| 12–15 | RCW3 | Register Control Word 0. Represents the value that is placed on MBA[1], MBA[0], MA[4], and MA[3] during writes to register control word 3. |
| 16–19 | RCW4 | Register Control Word 0. Represents the value that is placed on MBA[1], MBA[0], MA[4], and MA[3] during writes to register control word 4. |
| 20–23 | RCW5 | Register Control Word 0. Represents the value that is placed on MBA[1], MBA[0], MA[4], and MA[3] during writes to register control word 5. |
| 24–27 | RCW6 | Register Control Word 0. Represents the value that is placed on MBA[1], MBA[0], MA[4], and MA[3] during writes to register control word 6. |
| 28–31 | RCW7 | Register Control Word 0. Represents the value that is placed on MBA[1], MBA[0], MA[4], and MA[3] during writes to register control word 7. |

### 8.4.1.24 DDR SDRAM Register Control Word 2 (DDR_SDRAM_RCW_2)

The DDR Register Control Word 2register should be programmed with the intended values of the register control words if DDR_SDRAM_CFG[RCW_EN] is set. Each 4-bit field represents the value that is placed on MA[3], MA[4], MBA[0], and MBA[1] during register control word writes.

Offset 0x184 Access: Read/Write

| | 0 3 | 4 7 | 8 11 | 12 15 | 16 19 | 20 23 | 24 27 | 28 31 |
|---|---|---|---|---|---|---|---|---|
| R<br>W | RCW8 | RCW9 | RCW10 | RCW11 | RCW12 | RCW13 | RCW14 | RCW15 |
| Reset | All zeros | | | | | | | |

Figure 8-25. DDR Register Control Word 2 (DDR_SDRAM_RCW_2)

Table 8-28 describes the DDR_SDRAM_RCW_2 fields.

Table 8-30. DDR_Register Control Word 2 Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–3 | RCW8 | Register Control Word 0. Represents the value that is placed on MBA[1], MBA[0], MA[4], and MA[3] during writes to register control word 8. |
| 4–7 | RCW9 | Register Control Word 0. Represents the value that is placed on MBA[1], MBA[0], MA[4], and MA[3] during writes to register control word 9. |
| 8–11 | RCW10 | Register Control Word 0. Represents the value that is placed on MBA[1], MBA[0], MA[4], and MA[3] during writes to register control word 10. |

**Table 8-30. DDR_Register Control Word 2 Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 12–15 | RCW11 | Register Control Word 0. Represents the value that is placed on MBA[1], MBA[0], MA[4], and MA[3] during writes to register control word 11. |
| 16–19 | RCW12 | Register Control Word 0. Represents the value that is placed on MBA[1], MBA[0], MA[4], and MA[3] during writes to register control word 12. |
| 20–23 | RCW13 | Register Control Word 0. Represents the value that is placed on MBA[1], MBA[0], MA[4], and MA[3] during writes to register control word 13. |
| 24–27 | RCW14 | Register Control Word 0. Represents the value that is placed on MBA[1], MBA[0], MA[4], and MA[3] during writes to register control word 14. |
| 28–31 | RCW15 | Register Control Word 0. Represents the value that is placed on MBA[1], MBA[0], MA[4], and MA[3] during writes to register control word 15. |

### 8.4.1.25 DDR Debug Status Register 1 (DDRDSR_1)

The DDRDSR_1 register, shown in Figure 8-26, contains the DDR driver compensation input value and the current settings of the P and N FET impedance for MDIC*n*, command/control, and data.

Offset 0xB20 Access: Read only

| | 0 1 | 2 5 | 6 9 | 10 15 | 16 19 | 20 23 | 24 27 | 28 31 |
|---|---|---|---|---|---|---|---|---|
| R | DDRDC | MDICPZ | MDICNZ | — | CPZ | CNZ | DPZ | DNZ |
| W | | | | | | | | |
| Reset | All zeros | | | | | | | |

**Figure 8-26. DDR Debug Status Register 1 (DDRDSR_1)**

Table 8-31 describes the DDRDSR_1 fields.

**Table 8-31. DDRDSR_1 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–1 | DDRDC | DDR driver compensation input value |
| 2–5 | MDICPZ | Current setting of PFET driver MDIC impedance |
| 6–9 | MDICNZ | Current setting of NFET driver MDIC impedance |
| 10–15 | — | Reserved, should be cleared. |
| 16–19 | CPZ | Current setting of PFET driver command impedance |
| 20–23 | CNZ | Current setting of NFET driver command impedance |
| 24–27 | DPZ | Current setting of PFET driver data impedance |
| 28–31 | DNZ | Current setting of NFET driver data impedance |

### 8.4.1.26 DDR Debug Status Register 2 (DDRDSR_2)

The DDRDSR_2 register, shown in Figure 8-27, contains the current settings of the P and N FET impedance for the DDR drivers for clocks.

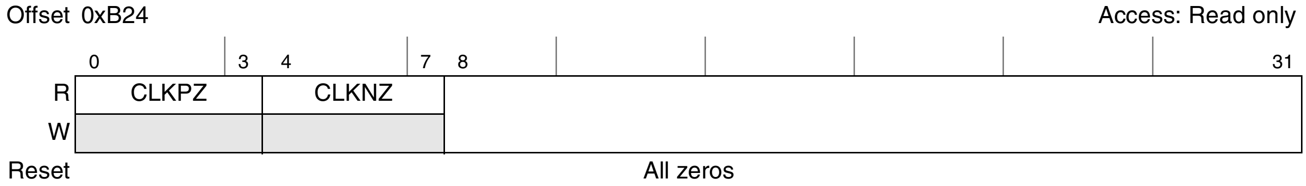


**Figure 8-27. DDR Debug Status Register 2 (DDRDSR_2)**

Table 8-32 describes the DDRDSR_2 fields.

**Table 8-32. DDRDSR_2 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–3 | CLKPZ | Current setting of PFET driver clock impedance |
| 4–7 | CLKNZ | Current setting of NFET driver clock impedance |
| 8–31 | — | Reserved |

### 8.4.1.27 DDR Control Driver Register 1 (DDRCDR_1)

DDRCDR_1, shown in Figure 8-28, sets the driver hardware compensation enable, the DDR MDIC driver P/N impedance, ODT termination value for IOs, driver software override enable for MDIC, driver software override enable for address/command, driver software override enable for data, the DDR address/command driver P/N impedance, and the DDR data driver P/N impedance.

The fields in DDRCDR_1, other than DDRCDR_1[ODT], are used to enable driver calibration with the MDIC[0:1] pins. This can be used to calibrate the DDR drivers to 18 ohms. However, this should only be used for full-strength driver applications.

Hardware DDR driver calibration is enabled by setting DDRCDR_1[DHC_EN].

**NOTE**

> All driver calibration, whether by software or hardware, should be done before the DDR controller is enabled (before DDR_SDRAM_CFG[MEM_EN] is set).

Software can be used to calibrate the drivers instead of the automatic hardware calibration. If software calibration is used, the following steps should be taken:

1. Set DDRCDR_1[DSO_MDIC_EN] and ensure that DDRCDR_1[DHC_EN] is cleared
2. Set the highest impedance (value 0000) for DDRCDR_1[DSO_MDICPZ]
3. Set DDRCDR_1[DSO_MDIC_PZ_OE] to enable the output enable for MDIC[0]
4. After at least 4 cycles, read DDRDSR_1[0]. If the value is 0, then use the next lowest impedance, and read DDRDSR_1[0] again. Once a value of 1 is detected, then leave DDRCDR_1[DSO_MDICPZ] at the calibrated value

5. Clear DDRCDR_1[DSO_MDIC_PZ_OE]
6. After DDRCDR_1[DSO_MDICPZ} is calibrated, set a value of 0000 for DDRCDR_1[DSO_MDICNZ]
7. Set DDRCDR_1[DSO_MDIC_NZ_OE] to enable the output enable for MDIC[1]
8. After at least 4 cycles, read DDRDSR_1[1]. If the value is 1, then use the next lowest impedance, and read DDRDSR_1[1] again. Once a value of 0 is detected, then leave DDRCDR_1[DSO_MDICNZ] at the calibrated value
9. Clear DDRCDR_1[DSO_MDIC_NZ_OE]

Note that the legal impedance values (from highest impedance to lowest impedance) for DDR2 (1.8 V) are:

- 0000
- 0001
- 0011
- 0010
- 0110
- 0111
- 0101
- 0100
- 1100
- 1101
- 1110
- 1010 (default full-strength impedance)
- 1011
- 1001

A value of 1111 provides the target for half-strength mode when driver calibration is not used.

Note that the legal impedance values (from highest impedance to lowest impedance) for DDR3 (1.5 V) are:

- 0000
- 0001
- 0011
- 0010
- 0110
- 0111 (default full-strength impedance)
- 0101
- 0100
- 1100
- 1101

A value of 0000 should be used for default half-strength mode when driver calibration is not used.

Note that the drivers may either be calibrated to full-strength or half-strength.

Offset 0xB28 Access: Read/Write

| | 0 | 1 | 2 5 | 6 9 | 10 | 11 | 12 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|
| R<br>W | DHC_EN | DSO_MDIC_EN | DSO_MDICPZ | DSO_MDICNZ | DSO_MDIC_PZ_OE | DSO_MDIC_NZ_OE | ODT | DSO_C_EN | DSO_D_EN |
| Reset | All zeros | | | | | | | | |

| | 16 19 | 20 23 | 24 27 | 28 31 |
|---|---|---|---|---|
| R<br>W | DSO_CPZ | DSO_CNZ | DSO_DPZ | DSO_DNZ |
| Reset | All zeros | | | |

**Figure 8-28. DDR Control Driver Register 1 (DDRCDR_1)**

Table 8-33 describes the DDRCDR_1 fields.

**Table 8-33. DDRCDR_1 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | DHC_EN | DDR driver hardware compensation enable |
| 1 | DSO_MDIC_EN | Driver software override enable for MDIC |
| 2–5 | DSO_MDICPZ | DDR driver software MDIC p-impedance override |
| 6–9 | DSO_MDICNZ | DDR driver software MDIC n-impedance override |
| 10 | DSO_MDIC_PZ_OE | Driver software override p-impedance output enable |
| 11 | DSO_MDIC_NZ_OE | Driver software override n-impedance output enable |
| 12–13 | ODT | ODT termination value for IOs. This field is combined with DDRCDR_2[ODT] to determine the termination value. Below is the termination based on concatenating these two fields.<br>000 75 Ω<br>001 55 Ω<br>010 60 Ω<br>011 50 Ω<br>100 150 Ω<br>101 43 Ω<br>110 120 Ω<br>111 Reserved<br>Note that the order of concatenation is (from left to right)<br>DDRCDR_1[ODT], DDRCDR_2[ODT] |
| 14 | DSO_C_EN | Driver software override enable for address/command |
| 15 | DSO_D_EN | Driver software override enable for data |
| 16–19 | DSO_CPZ | DDR driver software command p-impedance override |
| 20–23 | DSO_CNZ | DDR driver software command n-impedance override |
| 24–27 | DSO_DPZ | Driver software data p-impedance override |
| 28–31 | DSO_DNZ | Driver software data n-impedance override |

### 8.4.1.28 DDR Control Driver Register 2 (DDRCDR_2)

The DDRCDR_2, shown in Figure 8-29, sets the driver software override enable for clocks, and the DDR clocks driver P/N impedance.

Offset 0xB2C Access: Read/Write

| | 0 | 1–3 | 4–7 | 8–11 | 12–30 | 31 |
|---|---|---|---|---|---|---|
| R<br>W | DSO_CLK_EN | — | DSO_CLKPZ | DSO_CLKNZ | — | ODT |
| Reset | All zeros | | | | | |

**Figure 8-29. DDR Control Driver Register 2 (DDRCDR_2)**

Table 8-34 describes the DDRCDR_2 fields.

**Table 8-34. DDRCDR_2 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | DSO_CLK_EN | Driver software override enable for clocks |
| 1–3 | — | Reserved |
| 4–7 | DSO_CLKPZ | Driver software clocks p-impedance override |
| 8–11 | DSO_CLKNZ | Driver software clocks n-impedance override |
| 12–30 | — | Reserved |
| 31 | ODT | ODT termination value for IOs. This field is combined with DDRCDR_1[ODT] to determine the termination value. Below is the termination based on concatenating these two fields.<br>000 75 Ω<br>001 55 Ω<br>010 60 Ω<br>011 50 Ω<br>100 150 Ω<br>101 43 Ω<br>110 120 Ω<br>111 Reserved<br>Note that the order of concatenation is (from left to right)<br>DDRCDR_1[ODT], DDRCDR_2[ODT] |

### 8.4.1.29 DDR IP Block Revision 1 (DDR_IP_REV1)

The DDR IP block revision 1 register, shown in Figure 8-30, provides read-only fields with the IP block ID, along with major and minor revision information.

Offset 0xBF8 Access: Read Only

| | 0–15 | 16–23 | 24–31 |
|---|---|---|---|
| R | IP_ID | IP_MJ | IP_MN |
| W | | | |
| Reset | *n*[1] *n n n n n n n n n n n n n n n* | *n n n n n n n n* | *n n n n n n n n* |

**Figure 8-30. DDR IP Block Revision 1 (DDR_IP_REV1)**

[1] For reset values, see Table 8-35.

Table 8-35 describes the DDR_IP_REV1 fields.

Table 8-35. DDR_IP_REV1 Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–15 | IP_ID | IP block ID. For the DDR controller, this value is 0x0002. |
| 16–23 | IP_MJ | Major revision. This is currently set to 0x04. |

## 8.4.1.30 DDR IP Block Revision 2 (DDR_IP_REV2)

The DDR IP block revision 2 register, shown in Figure 8-31, provides read-only fields with the IP block integration and configuration options.

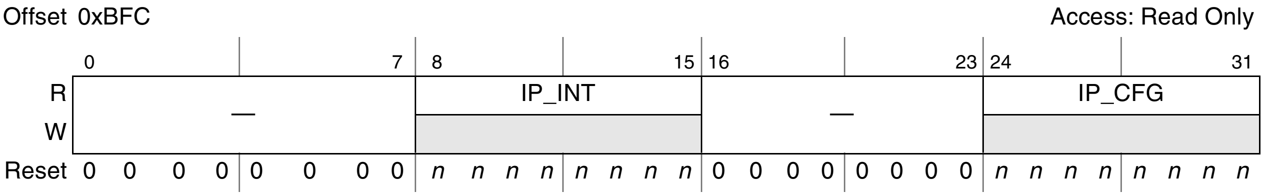

Offset 0xBFC
Access: Read Only
0
7
8
15
16
23
24
31
R
W
—
IP_INT
—
IP_CFG
Reset 0 0 0 0 0 0 0 0 n n n n n n n n 0 0 0 0 0 0 0 0 n n n n n n n n

Figure 8-31. DDR IP Block Revision 2 (DDR_IP_REV2)

Table 8-36 describes the DDR_IP_REV2 fields.

Table 8-36. DDR_IP_REV2 Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–7 | — | Reserved |
| 8–15 | IP_INT | IP block integration options |
| 16–23 | — | Reserved |
| 24–31 | IP_CFG | IP block configuration options |

## 8.4.1.31 Memory Data Path Error Injection Mask High (DATA_ERR_INJECT_HI)

The memory data path error injection mask high register is shown in Figure 8-32.

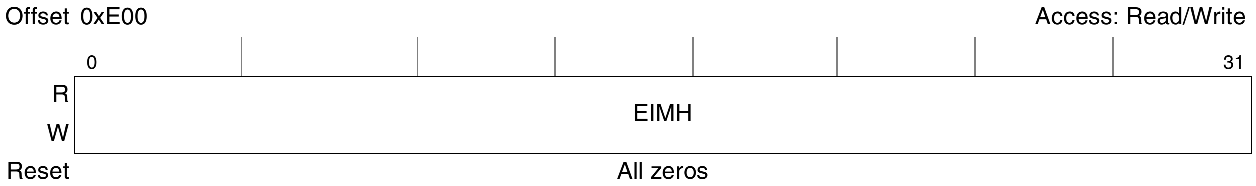


Figure 8-32. Memory Data Path Error Injection Mask High Register (DATA_ERR_INJECT_HI)

Table 8-37 describes the DATA_ERR_INJECT_HI fields.

Table 8-37. DATA_ERR_INJECT_HI Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–31 | EIMH | Error injection mask high data path. Used to test ECC by forcing errors on the high word of the data path. Setting a bit causes the corresponding data path bit to be inverted on memory bus writes. |

### 8.4.1.32 Memory Data Path Error Injection Mask Low (DATA_ERR_INJECT_LO)

The memory data path error injection mask low register is shown in Figure 8-33.

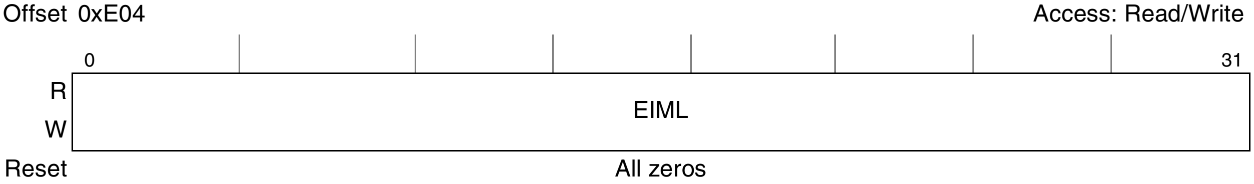


Figure 8-33. Memory Data Path Error Injection Mask Low Register (DATA_ERR_INJECT_LO)

Table 8-38 describes the DATA_ERR_INJECT_LO fields.

Table 8-38. DATA_ERR_INJECT_LO Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–31 | EIML | Error injection mask low data path. Used to test ECC by forcing errors on the low word of the data path. Setting a bit causes the corresponding data path bit to be inverted on memory bus writes. |

### 8.4.1.33 Memory Data Path Error Injection Mask ECC (ERR_INJECT)

The memory data path error injection mask ECC register, shown in Figure 8-34, sets the ECC mask, enables errors to be written to ECC memory, and allows the ECC byte to mirror the most significant data byte. In addition, a single address parity error may be injected through this register.

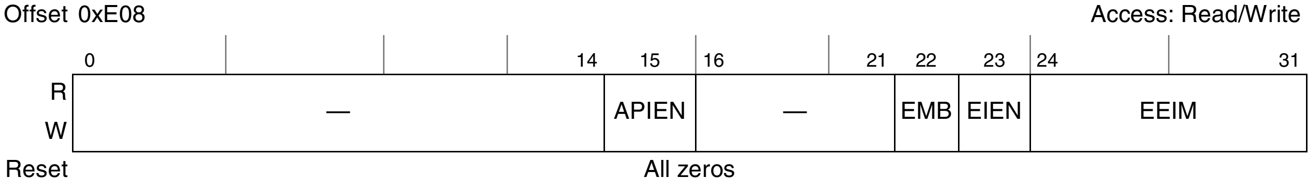


Figure 8-34. Memory Data Path Error Injection Mask ECC Register (ERR_INJECT)

Table 8-39 describes the ERR_INJECT fields.

Table 8-39. ERR_INJECT Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–14 | — | Reserved |
| 15 | APIEN | Address parity error injection enable. This bit is cleared by hardware after a single address parity error has been injected.<br>0 Address parity error injection disabled.<br>1 Address parity error injection enabled. |
| 16–21 | — | Reserved |
| 22 | EMB | ECC mirror byte<br>0 Mirror byte functionality disabled.<br>1 Mirror the most significant data path byte onto the ECC byte. |
| 23 | EIEN | Error injection enable<br>0 Error injection disabled.<br>1 Error injection enabled. This applies to the data mask bits, the ECC mask bits, and the ECC mirror bit. Note that error injection should not be enabled until the memory controller has been enabled through DDR_SDRAM_CFG[MEM_EN]. |
| 24–31 | EEIM | ECC error injection mask. Setting a mask bit causes the corresponding ECC bit to be inverted on memory bus writes. |

### 8.4.1.34 Memory Data Path Read Capture High (CAPTURE_DATA_HI)

The memory data path read capture high register, shown in Figure 8-35, stores the high word of the read data path during error capture.

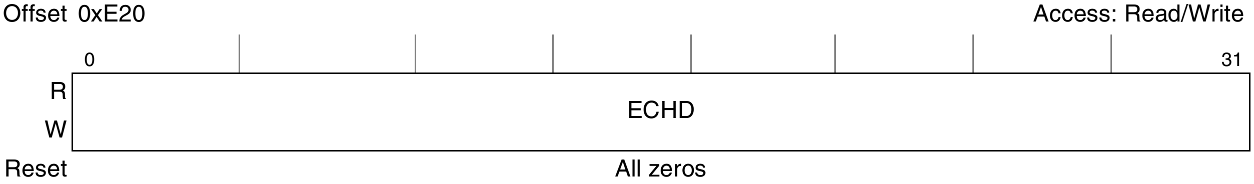


Figure 8-35. Memory Data Path Read Capture High Register (CAPTURE_DATA_HI)

Table 8-40 describes the CAPTURE_DATA_HI fields.

Table 8-40. CAPTURE_DATA_HI Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–31 | ECHD | Error capture high data path. Captures the high word of the data path when errors are detected. |

### 8.4.1.35 Memory Data Path Read Capture Low (CAPTURE_DATA_LO)

The memory data path read capture low register, shown in Figure 8-36, stores the low word of the read data path during error capture.

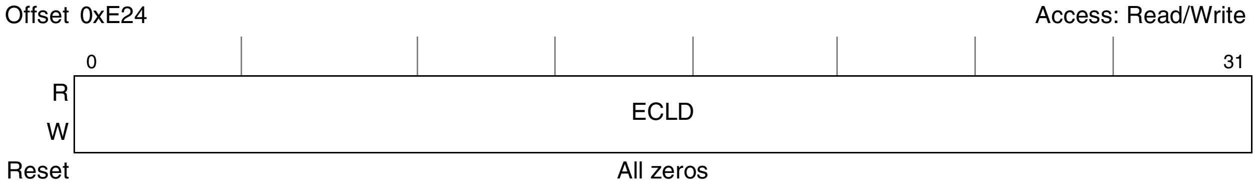


**Figure 8-36. Memory Data Path Read Capture Low Register (CAPTURE_DATA_LO)**

Table 8-41 describes the CAPTURE_DATA_LO fields.

**Table 8-41. CAPTURE_DATA_LO Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–31 | ECLD | Error capture low data path. Captures the low word of the data path when errors are detected. |

### 8.4.1.36 Memory Data Path Read Capture ECC (CAPTURE_ECC)

The memory data path read capture ECC register, shown in Figure 8-37, stores the ECC syndrome bits that were on the data bus when an error was detected.

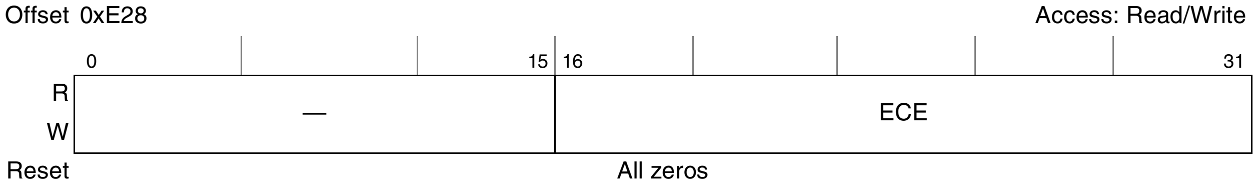


**Figure 8-37. Memory Data Path Read Capture ECC Register (CAPTURE_ECC)**

Table 8-42 describes the CAPTURE_ECC fields.

**Table 8-42. CAPTURE_ECC Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–15 | — | Reserved |
| 16–31 | ECE | Error capture ECC. Captures the ECC bits on the data path whenever errors are detected.<br>0:7—8-bit ECC for 1st 16 bits in 16-bit bus mode; should be ignored for 32-bit and 64-bit mode<br>8:15—8-bit ECC for 2nd 16 bits in 16-bit bus mode; 1st 32 bits in 32-bit bus mode; should be ignored for 64-bit bus mode<br>16:23—8-bit ECC for 3rd 16 bits in 16-bit bus mode; should be ignored for 32-bit and 64-bit mode<br>24:31—8-bit ECC for 4th 16 bits in 16-bit bus mode; 2nd 32 bits in 32-bit bus mode; all 64-bits in 64-bit bus mode |

### 8.4.1.37 Memory Error Detect (ERR_DETECT)

The memory error detect register stores the detection bits for multiple memory errors, single- and multiple-bit ECC errors, and memory select errors. It is a read/write register. A bit can be cleared by writing a one to the bit. System software can determine the type of memory error by examining the contents of this register. If an error is disabled with ERR_DISABLE, the corresponding error is never detected or captured in ERR_DETECT.

ERR_DETECT is shown in Figure 8-38.

Offset 0xE40 Access: w1c

| | 0 | 1 … 22 | 23 | 24 | 25 … 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|
| R | MME | — | APE | ACE | — | MBE | SBE | — | MSE |
| W | w1c | | w1c | w1c | | w1c | w1c | | w1c |
| Reset | All zeros | | | | | | | | |

**Figure 8-38. Memory Error Detect Register (ERR_DETECT)**

Table 8-43 describes the ERR_DETECT fields.

**Table 8-43. ERR_DETECT Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | MME | Multiple memory errors. This bit is cleared by software writing a 1.<br>0 Multiple memory errors of the same type were not detected.<br>1 Multiple memory errors of the same type were detected. |
| 1–22 | — | Reserved |
| 23 | APE | Address parity error. This bit is cleared by software writing a 1.<br>0 An address parity error has not been detected.<br>1 An address parity error has been detected. |
| 24 | ACE | Automatic calibration error. This bit is cleared by software writing a 1.<br>0 An automatic calibration error has not been detected.<br>1 An automatic calibration error has been detected. |
| 25–27 | — | Reserved |
| 28 | MBE | Multiple-bit error. This bit is cleared by software writing a 1.<br>0 A multiple-bit error has not been detected.<br>1 A multiple-bit error has been detected. |
| 29 | SBE | Single-bit ECC error. This bit is cleared by software writing a 1.<br>0 The number of single-bit ECC errors detected has not crossed the threshold set in ERR_SBE[SBET].<br>1 The number of single-bit ECC errors detected crossed the threshold set in ERR_SBE[SBET]. |
| 30 | — | Reserved |
| 31 | MSE | Memory select error. This bit is cleared by software writing a 1.<br>0 A memory select error has not been detected.<br>1 A memory select error has been detected. |

### 8.4.1.38 Memory Error Disable (ERR_DISABLE)

The memory error disable register, shown in Figure 8-39, allows selective disabling of the DDR controller’s error detection circuitry. Disabled errors are not detected or reported.

Offset 0xE44 Access: Read/Write

| | 0–22 | 23 | 24 | 25–27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|
| R<br>W | — | APED | ACED | — | MBED | SBED | — | MSED |
| Reset | All zeros | | | | | | | |

**Figure 8-39. Memory Error Disable Register (ERR_DISABLE)**

Table 8-44 describes the ERR_DISABLE fields.

**Table 8-44. ERR_DISABLE Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–22 | — | Reserved |
| 23 | APED | Address parity error disable<br>0 Address parity errors are detected if DDR_SDRAM_CFG_2[AP_EN] is set. They are reported if ERR_INT_EN[APEE] is set.<br>1 Address parity errors are not detected or reported. |
| 24 | ACED | Automatic calibration error disable<br>0 Automatic calibration errors are enabled.<br>1 Automatic calibration errors are disabled. |
| 25–27 | — | Reserved |
| 28 | MBED | Multiple-bit ECC error disable<br>0 Multiple-bit ECC errors are detected if DDR_SDRAM_CFG[ECC_EN] is set. They are reported if ERR_INT_EN[MBEE] is set. Note that uncorrectable read errors cause the assertion of *core_fault_in*, which causes the core to generate a machine check interrupt, unless it is disabled (by clearing HID1[RFXE]). If RFXE is zero and this error occurs, MBED and ERR_INT_EN[MBEE] must be zero and ECC_EN must be one to ensure that an interrupt is generated.<br>1 Multiple-bit ECC errors are not detected or reported. |
| 29 | SBED | Single-bit ECC error disable<br>0 Single-bit ECC errors are enabled.<br>1 Single-bit ECC errors are disabled. |
| 30 | — | Reserved |
| 31 | MSED | Memory select error disable<br>0 Memory select errors are enabled.<br>1 Memory select errors are disabled. |

### 8.4.1.39 Memory Error Interrupt Enable (ERR_INT_EN)

The memory error interrupt enable register, shown in Figure 8-40, enables ECC interrupts or memory select error interrupts. When an enabled interrupt condition occurs, the internal $\overline{int}$ signal is asserted to the programmable interrupt controller (PIC).

Offset 0xE48 Access: Read/Write

| | 0–22 | 23 | 24 | 25–27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|
| R<br>W | — | APEE | ACEE | — | MBEE | SBEE | — | MSEE |
| Reset | All zeros | | | | | | | |

**Figure 8-40. Memory Error Interrupt Enable Register (ERR_INT_EN)**

Table 8-45 describes the ERR_INT_EN fields.

**Table 8-45. ERR_INT_EN Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–22 | — | Reserved |
| 23 | APEE | Address parity error interrupt enable<br>0 Address parity errors cannot generate interrupts.<br>1 Address parity errors generate interrupts. |
| 24 | ACEE | Automatic calibration error interrupt enable<br>0 Automatic calibration errors cannot generate interrupts.<br>1 Automatic calibration errors generate interrupts. |
| 25–27 | — | Reserved |
| 28 | MBEE | Multiple-bit ECC error interrupt enable. Note that uncorrectable read errors may cause the assertion of *core_fault_in*, which causes the core to generate a machine check interrupt, unless it is disabled (by clearing HID1[RFXE]). If RFXE is zero and this error occurs, MBEE and ERR_DISABLE[MBED] must be zero and DDR_SDRAM_CFG[ECC_EN] must be set to ensure that an interrupt is generated. For more information, see Section 5.2, "e500 Core Integration and the Core Complex Bus (CCB)," and the *PowerPC™ e500 Core Family Reference Manual*.<br>0 Multiple-bit ECC errors cannot generate interrupts.<br>1 Multiple-bit ECC errors generate interrupts. |
| 29 | SBEE | Single-bit ECC error interrupt enable<br>0 Single-bit ECC errors cannot generate interrupts.<br>1 Single-bit ECC errors generate interrupts. |
| 30 | — | Reserved |
| 31 | MSEE | Memory select error interrupt enable<br>0 Memory select errors do not cause interrupts.<br>1 Memory select errors generate interrupts. |

### 8.4.1.40 Memory Error Attributes Capture (CAPTURE_ATTRIBUTES)

The memory error attributes capture register, shown in Figure 8-41, sets attributes for errors including type, size, source, and others.

Offset 0xE4C

Access: Read/Write

| | 0 | 1 3 | 4 | 5 7 | 8 10 | 11 15 | 16 17 | 18 19 | 20 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | BNUM | — | TSIZ | — | TSRC | — | TTYP | — | VLD |
| Reset | All zeros | | | | | | | | | |

**Figure 8-41. Memory Error Attributes Capture Register (CAPTURE_ATTRIBUTES)**

Table 8-46 describes the CAPTURE_ATTRIBUTES fields.

**Table 8-46. CAPTURE_ATTRIBUTES Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | — | Reserved |
| 1–3 | BNUM | Data beat number. Captures the doubleword number for the detected error. Relevant only for ECC errors. |
| 4 | — | Reserved |
| 5–7 | TSIZ | Transaction size for the error. Captures the transaction size in double words.<br>000 4 double words<br>001 1 double word<br>010 2 double words<br>011 3 double words<br>Others Reserved |
| 8–10 | — | Reserved |
| 16–17 | — | Reserved |
| 18–19 | TTYP | Transaction type for the error.<br>00 Reserved<br>01 Write<br>10 Read<br>11 Read-modify-write |
| 20–30 | — | Reserved |
| 31 | VLD | Valid. Set as soon as valid information is captured in the error capture registers. |

### 8.4.1.41 Memory Error Address Capture (CAPTURE_ADDRESS)

The memory error address capture register, shown in Figure 8-42, holds the 32 lsbs of a transaction when a DDR ECC error is detected.

Offset 0xE50

Access: Read/Write

| | 0 31 |
|---|---|
| R<br>W | CADDR |
| Reset | All zeros |

**Figure 8-42. Memory Error Address Capture Register (CAPTURE_ADDRESS)**

Table 8-47 describes the CAPTURE_ADDRESS fields.

**Table 8-47. CAPTURE_ADDRESS Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–31 | CADDR | Captured address. Captures the 32 lsbs of the transaction address when an error is detected. |

### 8.4.1.42 Memory Error Extended Address Capture (CAPTURE_EXT_ADDRESS)

The memory error extended address capture register, shown in Figure 8-43, holds the four most significant transaction bits when an error is detected.

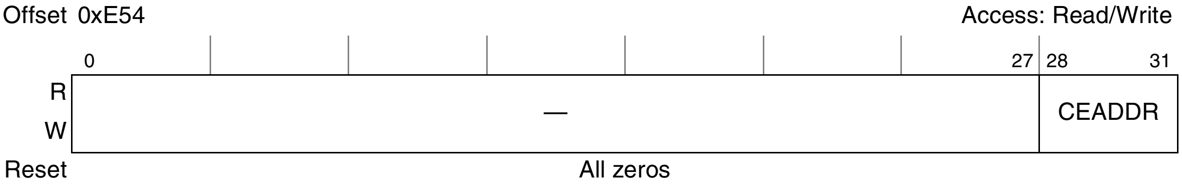


**Figure 8-43. Memory Error Extended Address Capture Register (CAPTURE_EXT_ADDRESS)**

Table 8-48 describes the CAPTURE_EXT_ADDRESS fields.

**Table 8-48. CAPTURE_EXT_ADDRESS Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–27 | — | Reserved |
| 28–31 | CEADDR | Captured extended address. Captures the 4 msbs of the transaction address when an error is detected |

### 8.4.1.43 Single-Bit ECC Memory Error Management (ERR_SBE)

The single-bit ECC memory error management register, shown in Figure 8-44, stores the threshold value for reporting single-bit errors and the number of single-bit errors counted since the last error report. When the counter field reaches the threshold, it wraps back to the reset value (0). If necessary, software must clear the counter after it has managed the error.

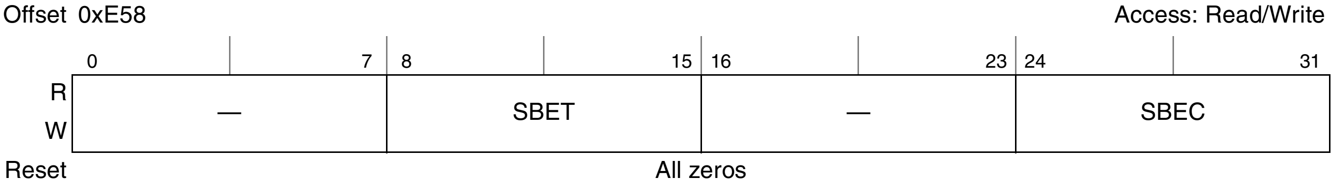


**Figure 8-44. Single-Bit ECC Memory Error Management Register (ERR_SBE)**

Table 8-49 describes the ERR_SBE fields.

Table 8-49. ERR_SBE Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–7 | — | Reserved |
| 8–15 | SBET | Single-bit error threshold. Establishes the number of single-bit errors that must be detected before an error condition is reported. |
| 16–23 | — | Reserved |
| 24–31 | SBEC | Single-bit error counter. Indicates the number of single-bit errors detected and corrected since the last error report. If single-bit error reporting is enabled, an error is reported and a machine check or critical interrupt is generated when this value equals SBET. SBEC is automatically cleared when the threshold value is reached. |

## 8.5 Functional Description

The DDR SDRAM controller controls processor and I/O interactions with system memory. It provides support for JEDEC-compliant DDR3 and DDR2 SDRAMs. The memory system allows a wide range of memory devices to be mapped to any arbitrary chip select, and support is provided for registered DIMMs and unbuffered DIMMs. However, registered DIMMs cannot be mixed with unbuffered DIMMs. In addition, DDR3 DIMM module specifications allow for vendors to use mirrored DIMMs, where some address and bank address lines are mirrored on the DIMM. The memory controller only supports these if the DDR_SDRAM_MD_CNTL register is used to initialize memory with DDR_SDRAM_CFG[BI] set.

Figure 8-45 is a high-level block diagram of the DDR memory controller. Requests are received from the internal mastering device and the address is decoded to generate the physical bank, logical bank, row, and column addresses. The transaction is compared with values in the row open table to determine if the address maps to an open page. If the transaction does not map to an open page, an active command is issued.

The memory interface supports as many as four physical banks of 64-/72-bit wide or 32-/40-bit wide memory. Bank sizes up to 4 Gbytes are supported, providing up to a maximum of 16 Gbytes of DDR main memory.

Programmable parameters allow for a variety of memory organizations and timings. Optional error checking and correcting (ECC) protection is provided for the DDR SDRAM data bus. Using ECC, the DDR memory controller detects and corrects all single-bit errors within the 64- or 32-bit data bus, detects all double-bit errors within the 64- or 32-bit data bus, and detects all errors within a nibble. The controller allows as many as 32 pages to be open simultaneously. The amount of time (in clock cycles) the pages remain open is programmable with DDR_SDRAM_INTERVAL[BSTOPRE].

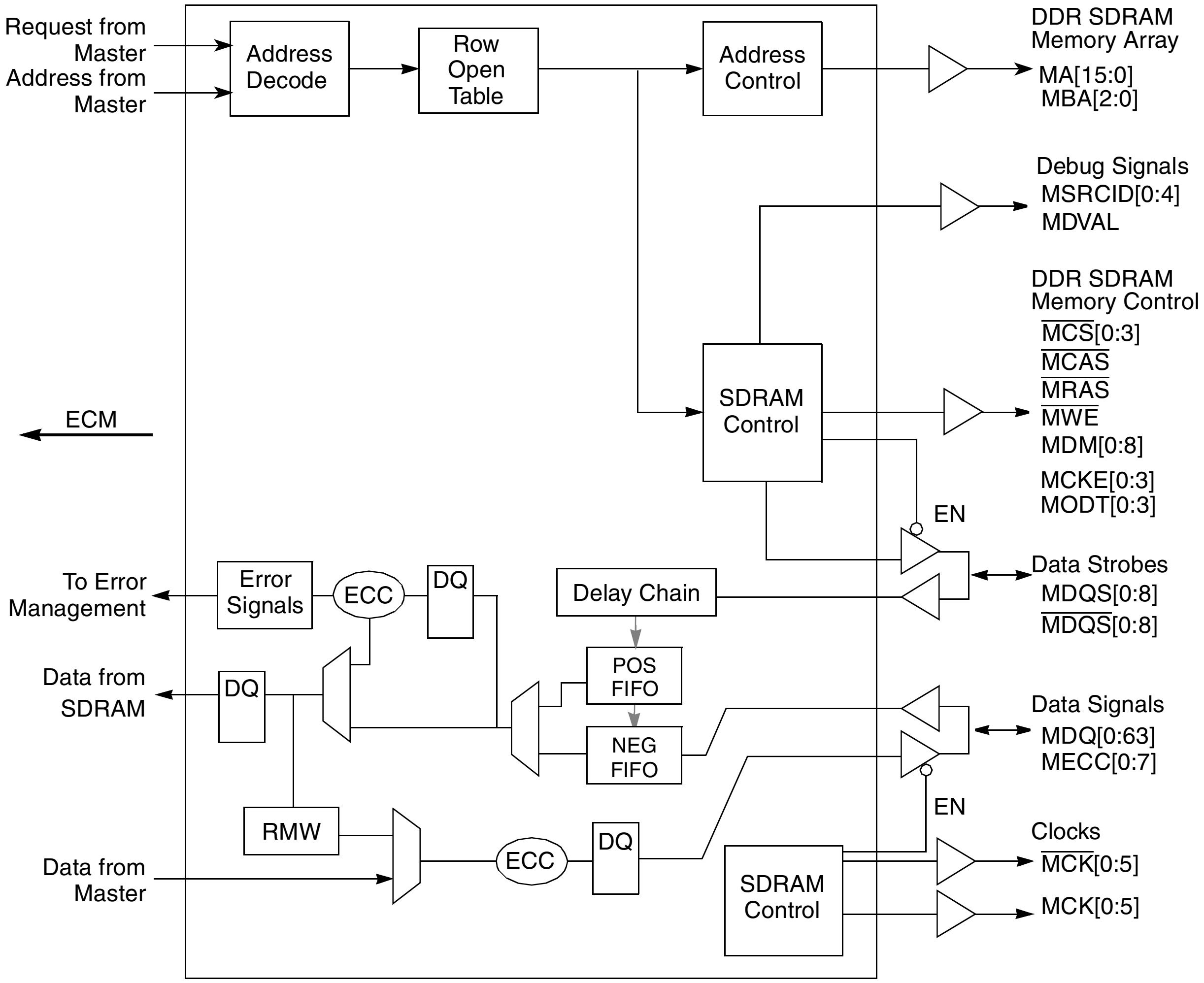


**Figure 8-45. DDR Memory Controller Block Diagram**

Read and write accesses to memory are burst oriented; accesses start at a selected location and continue for a programmed number of higher locations (4 or 8) in a programmed sequence. Accesses to closed pages start with the registration of an ACTIVE command followed by a READ or WRITE. (Accessing open pages does not require an ACTIVE command.) The address bits registered coincident with the activate command specifies the logical bank and row to be accessed. The address coincident with the READ or WRITE command specify the logical bank and starting column for the burst access.

The data interface is source synchronous, meaning whatever sources the data also provides a clocking signal to synchronize data reception. These bidirectional data strobes (MDQS[0:8]) are inputs to the controller during reads and outputs during writes. The DDR SDRAM specification requires the data strobe signals to be centered within the data tenure during writes and to be offset by the controller to the center of the data tenure during reads. This delay is implemented in the controller for both reads and writes.

When ECC is enabled, 1 clock cycle is added to the read path to check ECC and correct single-bit errors. ECC generation does not add a cycle to the write path.

The address and command interface is also source synchronous, although 1/8 cycle adjustments are provided for adjusting the clock alignment.

Figure 8-46 shows an example DDR SDRAM configuration with four logical banks.

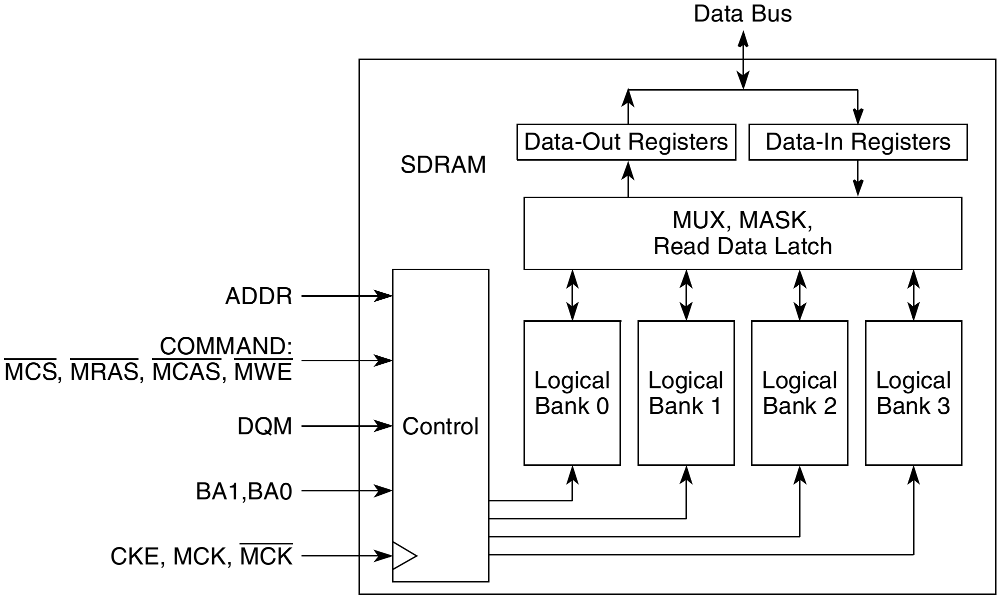


**Figure 8-46. Typical Dual Data Rate SDRAM Internal Organization**

Figure 8-47 shows some typical signal connections.

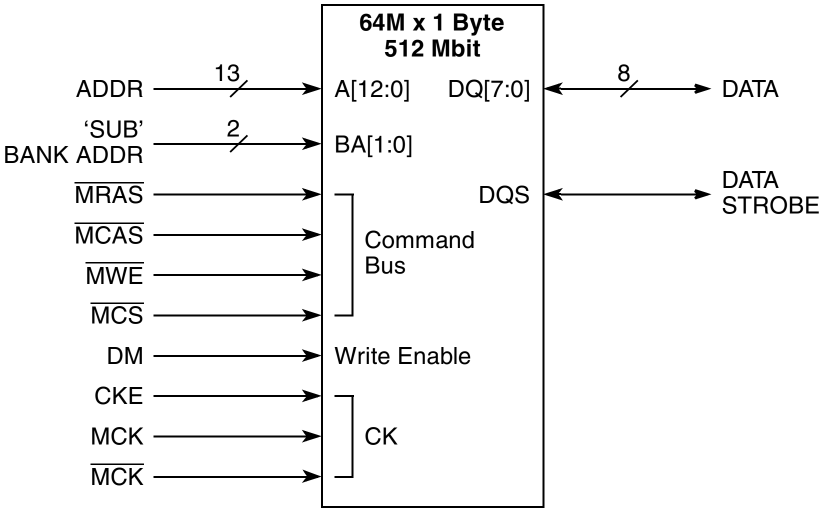
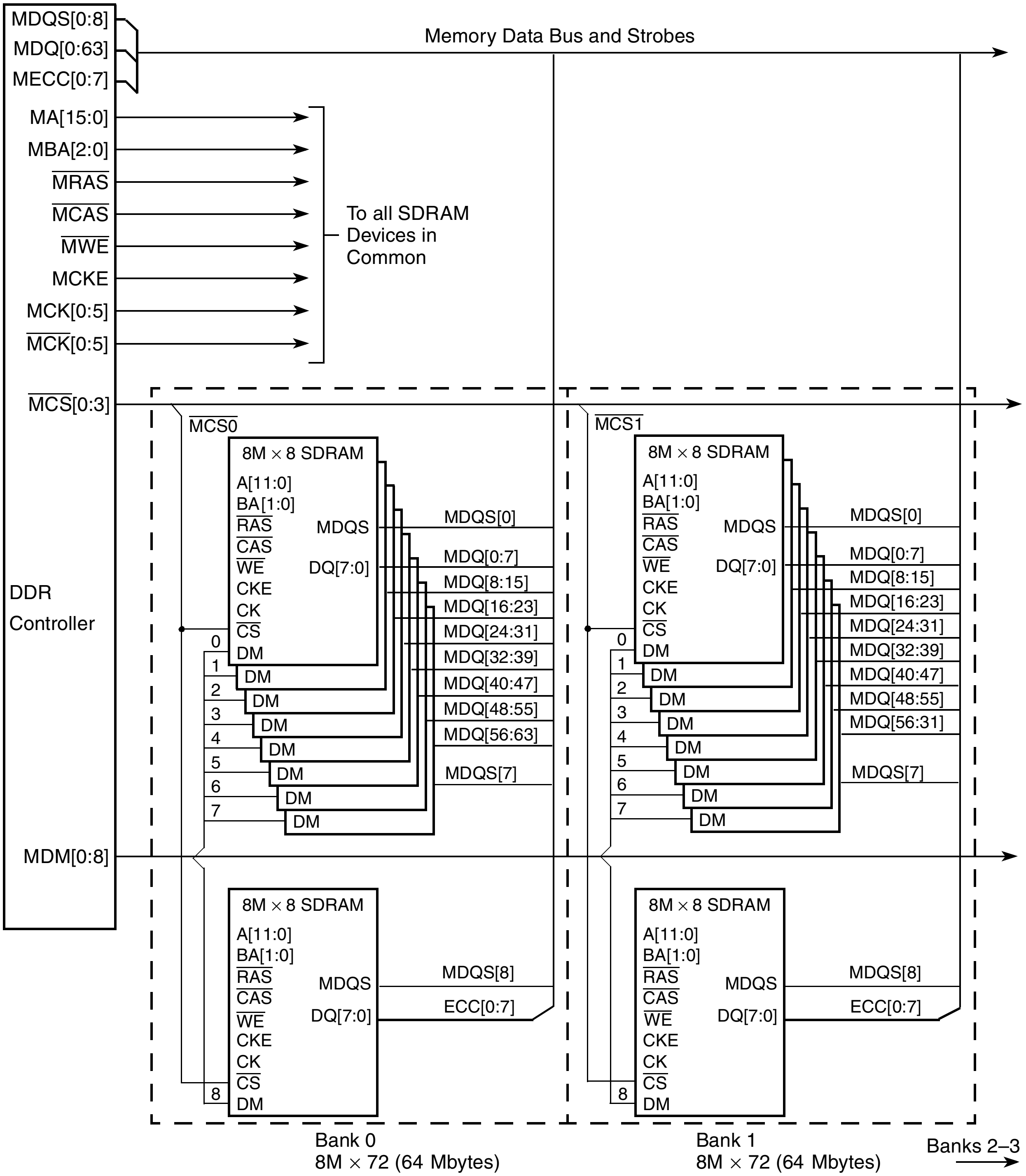


**Figure 8-47. Typical DDR SDRAM Interface Signals**

Figure 8-48 shows an example DDR SDRAM configuration with four physical banks each comprised of nine 8M × 8 DDR modules for a total of 256 Mbytes of system memory. One of the nine modules is used for the memory's ECC checking function. Certain address and control lines may require buffering. Analysis of the device's AC timing specifications, desired memory operating frequency, capacitive loads, and board routing loads can assist the system designer in deciding signal buffering requirements. The DDR memory controller drives 16 address pins, but in this example the DDR SDRAM devices use only 12 bits.



1. All signals are connected in common (in parallel) except for MCS[0:3], MCK[0:5], MDM[0:8], and the data bus signals.
2. Each of the MCS[0:3] signals correspond with a separate physical bank of memory.
3. Buffering may be needed if large memory arrays are used.
4. MCK[0:5] may be apportioned among all memory devices. Complementary bus is not shown.

**Figure 8-48. Example 256-Mbyte DDR SDRAM Configuration With ECC**

Section 8.5.12, “Error Management,” explains how the DDR memory controller handles errors.

### 8.5.1 DDR SDRAM Interface Operation

The DDR memory controller supports many different DDR SDRAM configurations. SDRAMs with different sizes can be used in the same system. Sixteen multiplexed address signals and three logical bank select signals support device densities from 64 Mbits to 4 Gbits. Four chip select ($\overline{\text{CS}}$) signals support up to two DIMMs of memory. The DDR SDRAM physical banks can be built from standard memory modules or directly-attached memory devices. The data path to individual physical banks is 64 or 32 bits wide, 72 or 40 bits with ECC. The DDR memory controller supports physical bank sizes from 16 Mbytes to 4 Gbytes. The physical banks can be constructed using x8, x16, or x32 memory devices. The memory technologies supported are 64 Mbits, 128 Mbits, 256 Mbits, 512 Mbits, 1 Gbit, 2 Gbits, and 4 Gbits. Nine data qualifier (DQM) signals provide byte selection for memory accesses.

**NOTE**

> An 8-bit DDR SDRAM device has a DQM signal and eight data signals (DQ[0:7]). A 16-bit DDR SDRAM device has two DQM signals associated with specific halves of the 16 data signals (DQ[0:7] and DQ[8:15]).

When ECC is enabled, all memory accesses are performed on double-word boundaries (that is, all DQM signals are set simultaneously). However, when ECC is disabled, the memory system uses the DQM signals for byte lane selection.

Table 8-50 shows the DDR memory controller’s relationships between data byte lane0–7, MDM[0:7], MDQS[0:7], and MDQ[0:63] when DDR SDRAM memories are used with x8 or x16 devices.

**Table 8-50. Byte Lane to Data Relationship**

| Data Byte Lane | Data Bus Mask | Data Bus Strobe | Data Bus 64-Bit Mode |
|---|---|---|---|
| 0 (MSB) | MDM[0] | MDQS[0] | MDQ[0:7] |
| 1 | MDM[1] | MDQS[1] | MDQ[8:15] |
| 2 | MDM[2] | MDQS[2] | MDQ[16:23] |
| 3 | MDM[3] | MDQS[3] | MDQ[24:31] |
| 4 | MDM[4] | MDQS[4] | MDQ[32:39] |
| 5 | MDM[5] | MDQS[5] | MDQ[40:47] |
| 6 | MDM[6] | MDQS[6] | MDQ[48:55] |
| 7 (LSB) | MDM[7] | MDQS[7] | MDQ[56:63] |

#### 8.5.1.1 Supported DDR SDRAM Organizations

Although the DDR memory controller multiplexes row and column address bits onto 16 memory address signals and 3 logical bank select signals, a physical bank may be implemented with memory devices requiring fewer than 31 address bits. The physical bank may be configured to provide from 12 to 16 row address bits, plus 2 or 3 logical bank-select bits and from 8–11 column address bits.

Table 8-52 and Table 8-53 describe DDR SDRAM device configurations supported by the DDR memory controller.

**NOTE**

DDR SDRAM is limited to 30 total address bits.

**Table 8-51. Supported DDR1 SDRAM Device Configurations**

| SDRAM Device | Device Configuration | Row x Column x Sub-Bank Bits | 64-Bit Bank Size | Four Banks of Memory |
|---|---|---|---|---|
| 64 Mbits | 8 Mbits x 8 | 12 x 9 x 2 | 64 Mbytes | 256 Mbytes |
| 64 Mbits[1] | 4 Mbits x 16 | 12 x 8 x 2 | 32 Mbytes | 128 Mbytes |
| 128 Mbits | 16 Mbits x 8 | 12 x 10 x 2 | 128 Mbytes | 512 Mbytes |
| 128 Mbits | 8 Mbits x 16 | 12 x 9 x 2 | 64 Mbytes | 256 Mbytes |
| 256 Mbits | 32 Mbits x 8 | 13 x 10 x 2 | 256 Mbytes | 1 Gbyte |
| 256 Mbits | 16 Mbits x 16 | 13 x 9 x 2 | 128 Mbytes | 512 Mbytes |
| 512 Mbits | 64 Mbits x 8 | 13 x 11 x 2 | 512 Mbytes | 2 Gbytes |
| 512 Mbits | 32 Mbits x 16 | 13 x 10 x 2 | 256 Mbytes | 1 Gbyte |
| 1 Gbit | 128 Mbits x 8 | 14 x 11 x 2 | 1 Gbyte | 4 Gbytes |
| 1 Gbit | 64 Mbits x 16 | 14 x 10 x 2 | 512 Mbytes | 2 Gbytes |
| 2 Gbits | 256 Mbits x 8 | 15 x 11 x 2 | 2 Gbytes | |
| 2 Gbits | 128 Mbits x 16 | 15 x 10 x 2 | 1 Gbyte | 4 Gbytes |
| 4 Gbits | 512 Mbits x 8 | 16 x 11 x 2 | 4 Gbytes | 16 Gbytes |
| 4 Gbits | 256 Mbits x 16 | 16 x 10 x 2 | 2 Gbytes | 8 Gbytes |

[1] This configuration is not supported in 16-bit bus mode.

**Table 8-52. Supported DDR2 SDRAM Device Configurations**

| SDRAM Device | Device Configuration | Row x Column x Sub-Bank Bits | 64-Bit Bank Size | Four Banks of Memory |
|---|---|---|---|---|
| 256 Mbits | 32 Mbits x 8 | 13 x 10 x 2 | 256 Mbytes | 1 Gbyte |
| 256 Mbits | 16 Mbits x 16 | 13 x 9 x 2 | 128 Mbytes | 512 Mbytes |
| 512 Mbits | 64 Mbits x 8 | 14 x 10 x 2 | 512 Mbytes | 2 Gbytes |
| 512 Mbits | 32 Mbits x 16 | 13 x 10 x 2 | 256 Mbytes | 1 Gbyte |
| 1 Gbit | 128 Mbits x 8 | 14 x 10 x 3 | 1 Gbyte | 4 Gbytes |
| 1 Gbit | 64 Mbits x 16 | 13 x 10 x 3 | 512 Mbytes | 2 Gbytes |
| 2 Gbits | 256 Mbits x 8 | 15 x 10 x 3 | 2 Gbytes | |
| 2 Gbits | 128Mbits x 16 | 14 x 10 x 3 | 1 Gbyte | 4 Gbytes |
| 4 Gbits | 512 Mbits x 8 | 16 x 10 x 3 | 4 Gbytes | 16 Gbytes |
| 4 Gbits | 256 Mbits x 16 | 15 x 10 x 3 | 2 Gbytes | |

Table 8-53. Supported DDR3 SDRAM Device Configurations

| SDRAM Device | Device Configuration | Row x Column x Sub-bank Bits | 64-Bit Bank Size | Four Banks of Memory |
|---|---|---|---|---|
| 512 Mbits | 64 Mbits x 8 | 13 x 10 x 3 | 512 Mbytes | 2 Gbytes |
| 512 Mbits | 32 Mbits x 16 | 12 x 10 x 2 | 256 Mbytes | 1 Gbyte |
| 1 Gbits | 128 Mbits x 8 | 14 x 10 x 3 | 1 Gbyte | 4 Gbytes |
| 1 Gbits | 64 Mbits x 16 | 13 x 10 x 3 | 512 Mbytes | 2 Gbytes |
| 2 Gbits | 256Mbits x 8 | 15 x 10 x 3 | 2 Gbytes | 8 Gbytes |
| 2 Gbits | 128Mbits x 16 | 14 x 10 x 3 | 1 Gbyte | 4 Gbytes |
| 4 Gbits | 512Mbits x 8 | 16 x 10 x 3 | 4 Gbytes | 16 Gbytes |
| 4 Gbits | 256Mbits x 16 | 15 x 10 x 3 | 2 Gbytes | 8 Gbytes |

Table 8-54. Supported DDR2 SDRAM Device Configurations—One Physical Bank

| SDRAM Device | Device Configuration | Row x Column x Sub-bank Bits | 16-Bit Bank Size |
|---|---|---|---|
| 256 Mbits | 32 Mbits x 8 | 13 x 10 x 2 | 64 Mbytes |
| 256 Mbits | 16 Mbits x 16 | 13 x 9 x 2 | 32 Mbytes |
| 512 Mbits | 64 Mbits x 8 | 14 x 10 x 2 | 128 Mbytes |
| 512 Mbits | 32 Mbits x 16 | 13 x 10 x 2 | 64 Mbytes |
| 1 Gbits | 128 Mbits x 8 | 14 x 10 x 3 | 256 Mbytes |
| 1 Gbits | 64 Mbits x 16 | 13 x 10 x 3 | 128 Mbytes |
| 2 Gbits | 256Mbits x 8 | 14 x 11 x 3 | 512 Mbytes |
| 2 Gbits | 128Mbits x 16 | 14 x 10 x 3 | 256 Mbytes |

If a transaction request is issued to the DDR memory controller and the address does not lie within any of the programmed address ranges for an enabled chip select, a memory select error is flagged. Errors are described in detail in Section 8.5.12, “Error Management.”

Using a memory-polling algorithm at power-on reset or by querying the JEDEC serial presence detect capability of memory modules, system firmware uses the memory-boundary registers to configure the DDR memory controller to map the size of each bank in memory. The memory controller uses its bank map to assert the appropriate $\overline{\text{MCS}}$*n* signal for memory accesses according to the provided bank starting and ending addresses. The memory banks are not required to be mapped to a contiguous address space.

## 8.5.2 DDR SDRAM Address Multiplexing

The following tables (, Table 8-55, Table 8-56) show the address bit encodings for each DDR SDRAM configuration. The address presented at the memory controller signals MA[15:0] use MA[15] as the msb and MA[0] as the lsb. Also, MA[10] is used as the auto-precharge bit in DDR2/DDR3 modes for reads and writes, so the column address can never use MA[10].

### Table 8-55. DDR2 Address Multiplexing for 64-Bit Data Bus with Interleaving and Partial Array Self Refresh Disabled

| Row x Col | | msb | Address from Core Master | | | | | | | | | | | | | | | | | | | | | | | | | | | | lsb |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33–35 |
| 16 x 10 x 3 | $\overline{\text{MRAS}}$ | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | | | | | | | | | |
| | MBA | | | | | | | | | | | | | | | | | 2 | 1 | 0 | | | | | | | | | | | |
| | $\overline{\text{MCAS}}$ | | | | | | | | | | | | | | | | | | | | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 15 x 10 x 3 | $\overline{\text{MRAS}}$ | | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | | | | | | | | | |
| | MBA | | | | | | | | | | | | | | | | | 2 | 1 | 0 | | | | | | | | | | | |
| | $\overline{\text{MCAS}}$ | | | | | | | | | | | | | | | | | | | | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 14 x 10 x 3 | $\overline{\text{MRAS}}$ | | | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | | | | | | | | | |
| | MBA | | | | | | | | | | | | | | | | | 2 | 1 | 0 | | | | | | | | | | | |
| | $\overline{\text{MCAS}}$ | | | | | | | | | | | | | | | | | | | | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 14 x 10 x 2 | $\overline{\text{MRAS}}$ | | | | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | | | | | | | | |
| | MBA | | | | | | | | | | | | | | | | | | 1 | 0 | | | | | | | | | | | |
| | $\overline{\text{MCAS}}$ | | | | | | | | | | | | | | | | | | | | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 13 x 10 x 3 | $\overline{\text{MRAS}}$ | | | | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | | | | | | | | | |
| | MBA | | | | | | | | | | | | | | | | | 2 | 1 | 0 | | | | | | | | | | | |
| | $\overline{\text{MCAS}}$ | | | | | | | | | | | | | | | | | | | | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 13 x 10 x 2 | $\overline{\text{MRAS}}$ | | | | | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | | | | | | | | |
| | MBA | | | | | | | | | | | | | | | | | | 1 | 0 | | | | | | | | | | | |
| | $\overline{\text{MCAS}}$ | | | | | | | | | | | | | | | | | | | | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 13 x 9 x 2 | $\overline{\text{MRAS}}$ | | | | | | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | | | | | | | |
| | MBA | | | | | | | | | | | | | | | | | | | 1 | 0 | | | | | | | | | | |
| | $\overline{\text{MCAS}}$ | | | | | | | | | | | | | | | | | | | | | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

### Table 8-56. DDR2 Address Multiplexing for 32-Bit Data Bus with Interleaving and Partial Array Self Refresh Disabled

| Row x Col | | msb | Address from Core Master | | | | | | | | | | | | | | | | | | | | | | | | | | | | | lsb |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34–35 |
| 16 x 10 x 3 | $\overline{\text{MRAS}}$ | | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | | | | | | | | | |
| | MBA | | | | | | | | | | | | | | | | | | 2 | 1 | 0 | | | | | | | | | | | |
| | $\overline{\text{MCAS}}$ | | | | | | | | | | | | | | | | | | | | | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 15 x 10 x 3 | $\overline{\text{MRAS}}$ | | | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | | | | | | | | | |
| | MBA | | | | | | | | | | | | | | | | | | 2 | 1 | 0 | | | | | | | | | | | |
| | $\overline{\text{MCAS}}$ | | | | | | | | | | | | | | | | | | | | | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

**Table 8-56. DDR2 Address Multiplexing for 32-Bit Data Bus with Interleaving and Partial Array Self Refresh Disabled (continued)**

| Row x Col | | msb | Address from Core Master | | | | | | | | | | | | | | | | | | | | | | | | | | | | lsb |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34–35 |
| 14 x 10 x 3 | $\overline{\text{MRAS}}$ | | | | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | | | | | | | | | |
| | MBA | | | | | | | | | | | | | | | | | | 2 | 1 | 0 | | | | | | | | | | | |
| | $\overline{\text{MCAS}}$ | | | | | | | | | | | | | | | | | | | | | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 14 x 10 x 2 | $\overline{\text{MRAS}}$ | | | | | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | | | | | | | | |
| | MBA | | | | | | | | | | | | | | | | | | | 1 | 0 | | | | | | | | | | | |
| | $\overline{\text{MCAS}}$ | | | | | | | | | | | | | | | | | | | | | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 13 x 10 x 3 | $\overline{\text{MRAS}}$ | | | | | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | | | | | | | | | |
| | MBA | | | | | | | | | | | | | | | | | | 2 | 1 | 0 | | | | | | | | | | | |
| | $\overline{\text{MCAS}}$ | | | | | | | | | | | | | | | | | | | | | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 13 x 10 x 2 | $\overline{\text{MRAS}}$ | | | | | | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | | | | | | | | |
| | MBA | | | | | | | | | | | | | | | | | | | 1 | 0 | | | | | | | | | | | |
| | $\overline{\text{MCAS}}$ | | | | | | | | | | | | | | | | | | | | | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 13 x 9 x 2 | $\overline{\text{MRAS}}$ | | | | | | | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | | | | | | | |
| | MBA | | | | | | | | | | | | | | | | | | | | 1 | 0 | | | | | | | | | | |
| | $\overline{\text{MCAS}}$ | | | | | | | | | | | | | | | | | | | | | | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

Chip select interleaving is supported for the memory controller, and is programmed in DDR_SDRAM_CFG[BA_INTLV_CTL]. Interleaving is supported between chip selects 0 and 1 or chip selects 2 and 3. In addition, interleaving between all four chip selects can be enabled. When interleaving is enabled, the chip selects being interleaved must use the same size of memory. If two chip selects are interleaved, then 1 extra bit in the address decode is used for the interleaving to determine which chip select to access. If four chip selects are interleaved, then two extra bits are required in the address decode.

Table 8-57 illustrates examples of address decode when interleaving between two chip selects, and Table 8-58 shows examples of address decode when interleaving between four chip selects.

**Table 8-57. Example of Address Multiplexing for 64-Bit Data Bus Interleaving between Two Banks with Partial Array Self Refresh Disabled**

| Row x Col | | msb | Address from Core Master | | | | | | | | | | | | | | | | | | | | | | | | | | | lsb |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33–35 |
| 14 x 10 x 3 | $\overline{\text{MRAS}}$ | | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | CS SEL | | | | | | | | | | | | | | |
| | MBA | | | | | | | | | | | | | | | | | 2 | 1 | 0 | | | | | | | | | | | |
| | $\overline{\text{MCAS}}$ | | | | | | | | | | | | | | | | | | | | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 14 x 10 x 2 | $\overline{\text{MRAS}}$ | | | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | CS SEL | | | | | | | | | | | | | |
| | MBA | | | | | | | | | | | | | | | | | | 1 | 0 | | | | | | | | | | | |
| | $\overline{\text{MCAS}}$ | | | | | | | | | | | | | | | | | | | | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

Table 8-57. Example of Address Multiplexing for 64-Bit Data Bus Interleaving between Two Banks with Partial Array Self Refresh Disabled (continued)

| Row x Col | | msb | Address from Core Master | | | | | | | | | | | | | | | | | | | | | | | | | | | | lsb |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33–35 |
| 13 x 10 x 3 | $\overline{\text{MRAS}}$ | | | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | CS SEL | | | | | | | | | | | | | | |
| | MBA | | | | | | | | | | | | | | | | | 2 | 1 | 0 | | | | | | | | | | | |
| | $\overline{\text{MCAS}}$ | | | | | | | | | | | | | | | | | | | | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 13 x 10 x 2 | $\overline{\text{MRAS}}$ | | | | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | CS SEL | | | | | | | | | | | | | |
| | MBA | | | | | | | | | | | | | | | | | | 1 | 0 | | | | | | | | | | | |
| | $\overline{\text{MCAS}}$ | | | | | | | | | | | | | | | | | | | | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

Table 8-58. Example of Address Multiplexing for 64-Bit Data Bus Interleaving between Four Banks with Partial Array Self Refresh Disabled

| Row x Col | | msb | Address from Core Master | | | | | | | | | | | | | | | | | | | | | | | | | | | | lsb |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33–35 |
| 14 x 10 x 3 | $\overline{\text{MRAS}}$ | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | CS SEL | | | | | | | | | | | | | | | |
| | MBA | | | | | | | | | | | | | | | | | 2 | 1 | 0 | | | | | | | | | | | |
| | $\overline{\text{MCAS}}$ | | | | | | | | | | | | | | | | | | | | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 14 x 10 x 2 | $\overline{\text{MRAS}}$ | | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | CS SEL | | | | | | | | | | | | | | |
| | MBA | | | | | | | | | | | | | | | | | | 1 | 0 | | | | | | | | | | | |
| | $\overline{\text{MCAS}}$ | | | | | | | | | | | | | | | | | | | | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 13 x 10 x 3 | $\overline{\text{MRAS}}$ | | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | CS SEL | | | | | | | | | | | | | | | |
| | MBA | | | | | | | | | | | | | | | | | 2 | 1 | 0 | | | | | | | | | | | |
| | $\overline{\text{MCAS}}$ | | | | | | | | | | | | | | | | | | | | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 13 x 10 x 2 | $\overline{\text{MRAS}}$ | | | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | CS SEL | | | | | | | | | | | | | | |
| | MBA | | | | | | | | | | | | | | | | | | 1 | 0 | | | | | | | | | | | |
| | $\overline{\text{MCAS}}$ | | | | | | | | | | | | | | | | | | | | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

Partial Array Self Refresh (PASR) can be enabled for any chip select using the CS*n*_CONFIG_2[PASR_CFG] fields. If PASR is enabled for a given chip select, then the sub-bank and row decode is swapped, and the sub-bank is decoded as the most significant portion of the DRAM address, as shown in Table 8-59. If chip select interleaving and PASR are enabled for a chip select, then the interleaved chip select bit is placed immediately to the left of the column decode, as shown in Table 8-60.

**Table 8-59. DDR2 Address Multiplexing with Partial Array Self Refresh Enabled**

| Row x Col | | msb | Address from Core Master | | | | | | | | | | | | | | | | | | | | | | | | | | | | | lsb |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34–35 |
| 16 x 10 x 3 | $\overline{\text{MRAS}}$ | | | | | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | | | | | | |
| | MBA | | 2 | 1 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | $\overline{\text{MCAS}}$ | | | | | | | | | | | | | | | | | | | | | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 15 x 10 x 3 | $\overline{\text{MRAS}}$ | | | | | | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | | | | | | |
| | MBA | | | 2 | 1 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | $\overline{\text{MCAS}}$ | | | | | | | | | | | | | | | | | | | | | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 14 x 10 x 3 | $\overline{\text{MRAS}}$ | | | | | | | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | | | | | | |
| | MBA | | | | 2 | 1 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | |
| | $\overline{\text{MCAS}}$ | | | | | | | | | | | | | | | | | | | | | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 14 x 10 x 2 | $\overline{\text{MRAS}}$ | | | | | | | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | | | | | | |
| | MBA | | | | | 1 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | |
| | $\overline{\text{MCAS}}$ | | | | | | | | | | | | | | | | | | | | | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 13 x 10 x 3 | $\overline{\text{MRAS}}$ | | | | | | | | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | | | | | | |
| | MBA | | | | | 2 | 1 | 0 | | | | | | | | | | | | | | | | | | | | | | | | |
| | $\overline{\text{MCAS}}$ | | | | | | | | | | | | | | | | | | | | | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 13 x 10 x 2 | $\overline{\text{MRAS}}$ | | | | | | | | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | | | | | | |
| | MBA | | | | | | 1 | 0 | | | | | | | | | | | | | | | | | | | | | | | | |
| | $\overline{\text{MCAS}}$ | | | | | | | | | | | | | | | | | | | | | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 13 x 9 x 2 | $\overline{\text{MRAS}}$ | | | | | | | | | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | | | | | |
| | MBA | | | | | | | 1 | 0 | | | | | | | | | | | | | | | | | | | | | | | |
| | $\overline{\text{MCAS}}$ | | | | | | | | | | | | | | | | | | | | | | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

**Table 8-60. Example of Address Multiplexing for 64-bit Data Bus Interleaving Between Two Banks with Partial Array Self Refresh Enabled**

| Row x Col | | msb | | Address from Core Master | | | | | | | | | | | | | | | | | | | | | | | | | | | lsb |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33–35 |
| 14 x 10 x3 | $\overline{\text{MRAS}}$ | | | | | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | CS SEL | | | | | | | | | | | |
| | MBA | | 2 | 1 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | $\overline{\text{MCAS}}$ | | | | | | | | | | | | | | | | | | | | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 14 x 10 x 2 | $\overline{\text{MRAS}}$ | | | | | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | CS SEL | | | | | | | | | | | |
| | MBA | | | 1 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | $\overline{\text{MCAS}}$ | | | | | | | | | | | | | | | | | | | | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

**Table 8-60. Example of Address Multiplexing for 64-bit Data Bus Interleaving Between Two Banks with Partial Array Self Refresh Enabled (continued)**

| Row x Col | | msb | | Address from Core Master | | | | | | | | | | | | | | | | | | | | | | | | | | | lsb |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33–35 |
| 13 x 10 x 3 | $\overline{\text{MRAS}}$ | | | | | | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | CS SEL | | | | | | | | | | | |
| | MBA | | | 2 | 1 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | |
| | $\overline{\text{MCAS}}$ | | | | | | | | | | | | | | | | | | | | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 13 x 10 x 2 | $\overline{\text{MRAS}}$ | | | | | | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | CS SEL | | | | | | | | | | | |
| | MBA | | | | 1 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | |
| | $\overline{\text{MCAS}}$ | | | | | | | | | | | | | | | | | | | | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

## 8.5.3 JEDEC Standard DDR SDRAM Interface Commands

The following section describes the commands and timings the controller uses when operating in DDR3 or DDR2 modes.

All read or write accesses to DDR SDRAM are performed by the DDR memory controller using JEDEC standard DDR SDRAM interface commands. The SDRAM device samples command and address inputs on rising edges of the memory clock; data is sampled using both the rising and falling edges of DQS. Data read from the DDR SDRAM is also sampled on both edges of DQS.

The following DDR SDRAM interface commands (summarized in Table 8-61) are provided by the DDR controller. All actions for these commands are described from the perspective of the SDRAM device.

- Row activate—Latches row address and initiates memory read of that row. Row data is latched in SDRAM sense amplifiers and must be restored by a precharge command before another row activate occurs.
- Precharge—Restores data from the sense amplifiers to the appropriate row. Also initializes the sense amplifiers in preparation for reading another row in the memory array (performing another activate command). Precharge must occur after read or write, if the row address changes on the next open page mode access.
- Read—Latches column address and transfers data from the selected sense amplifier to the output buffer as determined by the column address. During each succeeding clock edge, additional data is driven without additional read commands. The amount of data transferred is determined by the burst size which defaults to 4.
- Write—Latches column address and transfers data from the data pins to the selected sense amplifier as determined by the column address. During each succeeding clock edge, additional data is transferred to the sense amplifiers from the data pins without additional write commands. The amount of data transferred is determined by the data masks and the burst size, which is set to four by the DDR memory controller.
- Refresh (similar to $\overline{\text{MCAS}}$ before $\overline{\text{MRAS}}$)—Causes a row to be read in all logical banks (JEDEC SDRAM) as determined by the refresh row address counter. This refresh row address counter is internal to the SDRAM. After being read, the row is automatically rewritten in the memory array. All logical banks must be in a precharged state before executing a refresh. The memory controller

also supports posted refreshes, where several refreshes may be executed at once, and the refresh interval may be extended.

- Mode register set (for configuration)—Allows setting of DDR SDRAM options. These options are: $\overline{\text{MCAS}}$ latency, additive latency (for DDR2), write recovery (for DDR2), burst type, and burst length. $\overline{\text{MCAS}}$ latency may be chosen as provided by the preferred SDRAM (some SDRAMs provide $\overline{\text{MCAS}}$ latency {1,2,3}, some provide $\overline{\text{MCAS}}$ latency {1,2,3,4,5}, and so on). Burst type is always sequential. Although some SDRAMs provide burst lengths of 1, 2, 4, 8, and page size, this memory controller supports a burst length of 4. A burst length of 8 is supported for DDR3memory only. For DDR2 in 32-bit bus mode, all 32-byte burst accesses from the platform are split into two 16-byte (that is, 4-beat) accesses to the SDRAMs in the memory controller. The mode register set command is performed by the DDR memory controller during system initialization. Parameters such as mode register data, $\overline{\text{MCAS}}$ latency, burst length, and burst type, are set by software in DDR_SDRAM_MODE[SDMODE] and transferred to the SDRAM array by the DDR memory controller after DDR_SDRAM_CFG[MEM_EN] is set. If DDR_SDRAM_CFG[BI] is set to bypass the automatic initialization, then the MODE registers can be configured through software through use of the DDR_SDRAM_MD_CNTL register.
- Self refresh (for long periods of standby)—Used when the device is in standby for very long periods of time. Automatically generates internal refresh cycles to keep the data in all memory banks refreshed. Before execution of this command, the DDR controller places all logical banks in a precharged state.

**Table 8-61. DDR SDRAM Command Table**

| Operation | CKE Prev. | CKE Current | $\overline{\text{MCS}}$ | $\overline{\text{MRAS}}$ | $\overline{\text{MCAS}}$ | $\overline{\text{MWE}}$ | MBA | MA10 | MA |
|---|---|---|---|---|---|---|---|---|---|
| Activate | H | H | L | L | H | H | Logical bank select | Row | Row |
| Precharge select logical bank | H | H | L | L | H | L | Logical bank select | L | X |
| Precharge all logical banks | H | H | L | L | H | L | X | H | X |
| Read | H | H | L | H | L | H | Logical bank select | L | Column |
| Read with auto-precharge | H | H | L | H | L | H | Logical bank select | H | Column |
| Write | H | H | L | H | L | L | Logical bank select | L | Column |
| Write with auto-precharge | H | H | L | H | L | L | Logical bank select | H | Column |
| Mode register set | H | H | L | L | L | L | Opcode | Opcode | Opcode and mode |
| Auto refresh | H | H | L | L | L | H | X | X | X |
| Self refresh | H | L | L | L | L | H | X | X | X |

## 8.5.4 DDR SDRAM Interface Timing

The DDR memory controller supports four-beat bursts to SDRAM. For single-beat reads, the DDR memory controller performs a four- (or eight-) beat burst read, but ignores the last three (or seven) beats.

Single-beat writes are performed by masking the last three (or seven) beats of the four- (or eight-) beat burst using the data mask MDM[0:8]. If ECC is disabled, writes smaller than double words are performed by appropriately activating the data mask. If ECC is enabled, the controller performs a read-modify write.

### NOTE

If a second read or write is pending, reads shorter than four beats are not terminated early even if some data is irrelevant.

To accommodate available memory technologies across a wide spectrum of operating frequencies, the DDR memory controller allows the setting of the intervals defined in Table 8-62 with granularity of one memory clock cycle, except for CASLAT, which can be programmed with 1/2 clock granularity.

**Table 8-62. DDR SDRAM Interface Timing Intervals**

| Timing Intervals | Definition |
|---|---|
| ACTTOACT | The number of clock cycles from a bank-activate command until another bank-activate command within a physical bank. This interval is listed in the AC specifications of the SDRAM as $t_{RRD}$. |
| ACTTOPRE | The number of clock cycles from an activate command until a precharge command is allowed. This interval is listed in the AC specifications of the SDRAM as $t_{RAS}$. |
| ACTTORW | The number of clock cycles from an activate command until a read or write command is allowed. This interval is listed in the AC specifications of the SDRAM as $t_{RCD}$. |
| BSTOPRE | The number of clock cycles to maintain a page open after an access. The page open duration counter is reloaded with BSTOPRE each time the page is accessed (including page hits). When the counter expires, the open page is closed with an SDRAM precharge bank command as soon as possible. |
| CASLAT | Used in conjunction with additive latency to obtain the READ latency. The number of clock cycles between the registration of a READ command by the SDRAM and the availability of the first piece of output data. If a READ command is registered at clock edge *n*, and the read latency is *m* clocks, the data is available nominally coincident with clock edge $n + m$. |
| PRETOACT | The number of clock cycles from a precharge command until an activate or a refresh command is allowed. This interval is listed in the AC specifications of the SDRAM as $t_{RP}$ |
| REFINT | Refresh interval. Represents the number of memory bus clock cycles between refresh cycles. Depending on DDR_SDRAM_CFG_2[NUM_PR], some number of rows are refreshed in each SDRAM bank during each refresh cycle. The value of REFINT depends on the specific SDRAMs used and the frequency of the interface as $t_{RP}$. |
| REFREC | The number of clock cycles from the refresh command until an activate command is allowed. This can be calculated by referring to the AC specification of the SDRAM device. The AC specification indicates a maximum refresh-to-activate interval in nanoseconds. |
| WR_DATA_DELAY | Provides different options for the timing between a write command and the write data strobe. This allows write data to be sent later than the nominal time to meet the SDRAM timing requirement between the registration of a write command and the reception of a data strobe associated with the write command. The specification dictates that the data strobe may not be received earlier than 75% of a cycle, or later than 125% of a cycle, from the registration of a write command. This parameter is not defined in the SDRAM specification. It is implementation-specific, defined for the DDR memory controller in TIMING_CFG_2. |
| WRREC | The number of clock cycles from the last beat of a write until a precharge command is allowed. This interval, write recovery time, is listed in the AC specifications of the SDRAM as $t_{WR}$. |
| WRTORD | Last write pair to read command. Controls the number of clock cycles from the last write data pair to the subsequent read command to the same bank as $t_{WTR}$. |

The value of the above parameters (in whole clock cycles) must be set by boot code at system start-up (in the TIMING_CFG_0, TIMING_CFG_1, TIMING_CFG_2, and TIMING_CFG_3 registers as described in Section 8.4.1.5, "DDR SDRAM Timing Configuration 0 (TIMING_CFG_0)," Section 8.4.1.6, "DDR SDRAM Timing Configuration 1 (TIMING_CFG_1)," Section 8.4.1.7, "DDR SDRAM Timing Configuration 2 (TIMING_CFG_2)," and Section 8.4.1.4, "DDR SDRAM Timing Configuration 3 (TIMING_CFG_3)") and be kept in the DDR memory controller configuration register space.

The following figures show SDRAM timing for various types of accesses. System software is responsible (at reset) for optimally configuring SDRAM timing parameters. The programmable timing parameters apply to both read and write timing configuration. The configuration process must be completed and the DDR SDRAM initialized before any accesses to SDRAM are attempted.

Figure 8-49 through Figure 8-51 show DDR SDRAM timing for various types of accesses; see Figure 8-49 for a single-beat read operation, Figure 8-50 for a single-beat write operation, and Figure 8-51 for a double word write operation. Note that all signal transitions occur on the rising edge of the memory bus clock and that single-beat read operations are identical to burst-reads. These figures assume the CLK_ADJUST is set to 1/2 DRAM cycle, an additive latency of 0 DRAM cycles is used, and the write latency is 1 DRAM cycle.

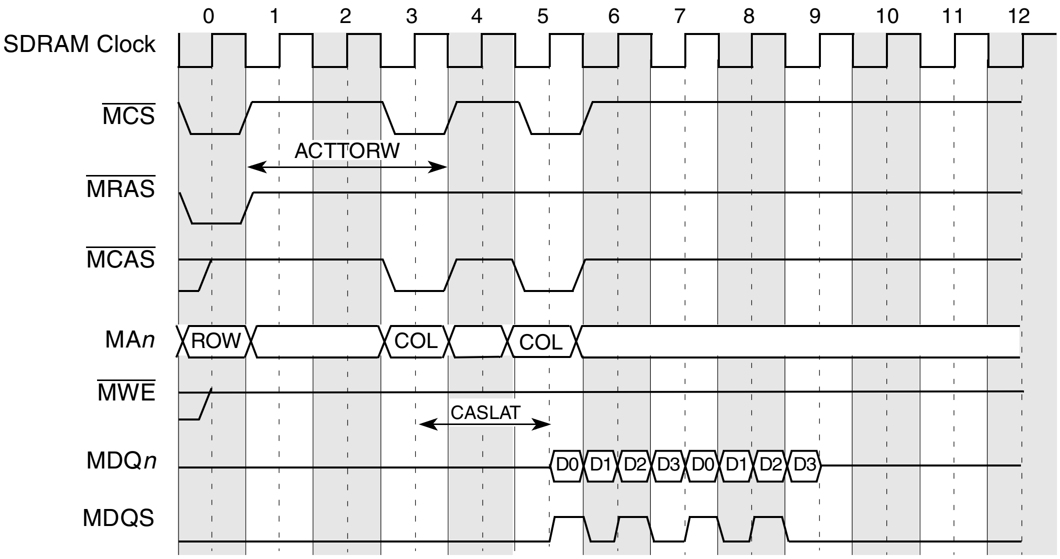


**Figure 8-49. DDR SDRAM Burst Read Timing—ACTTORW = 3, $\overline{\text{MCAS}}$ Latency = 2**

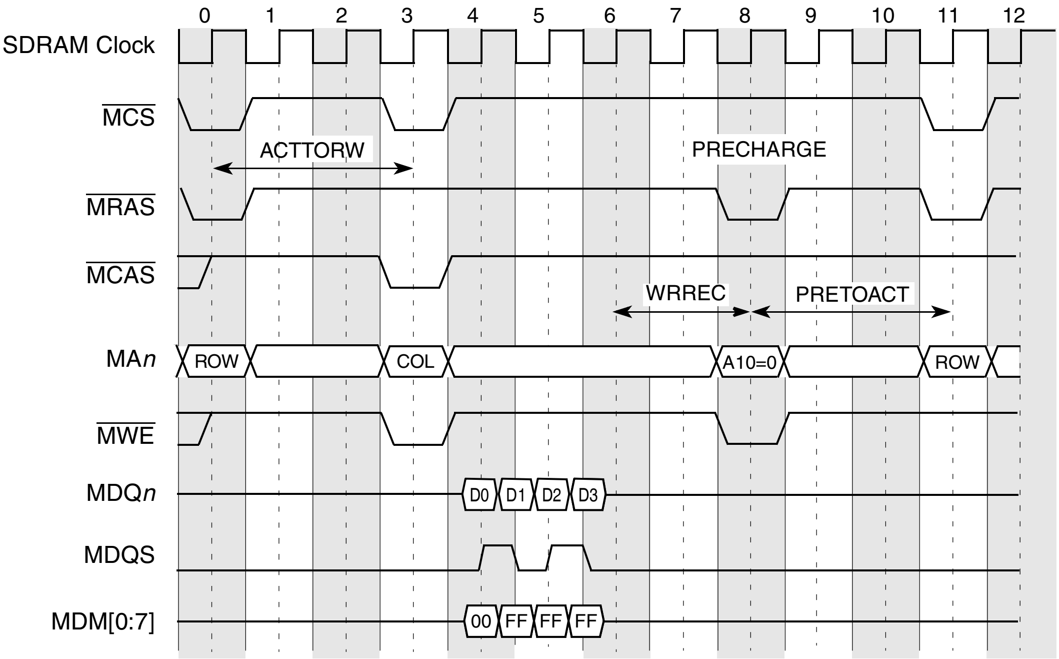


**Figure 8-50. DDR SDRAM Single-Beat (Double Word) Write Timing—ACTTOR**

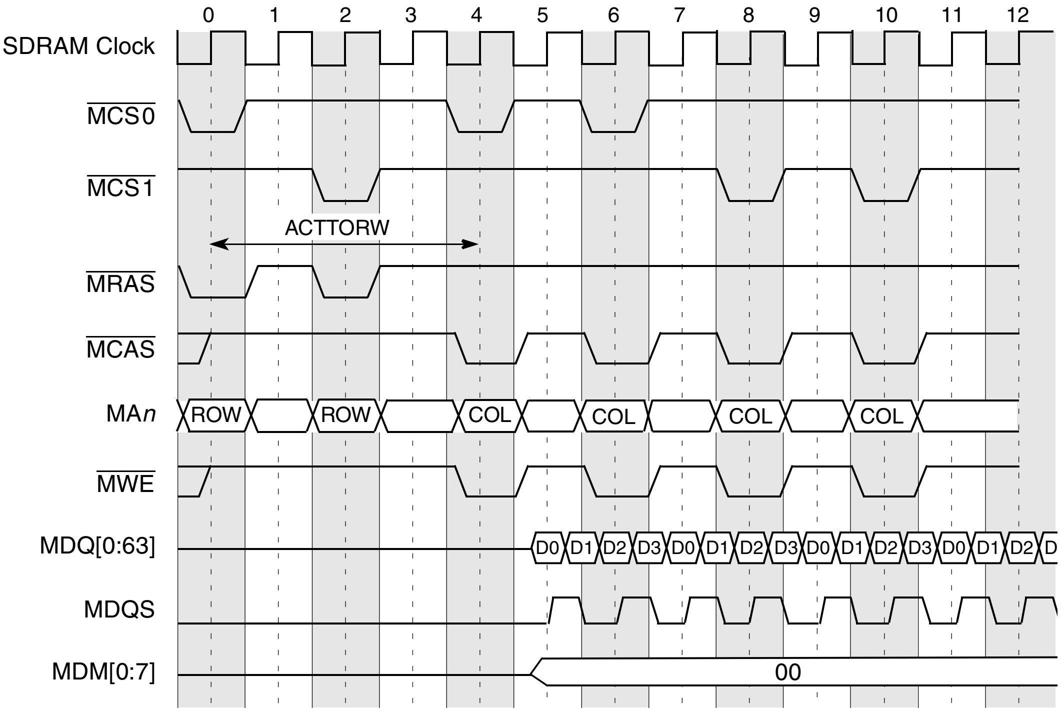


**Figure 8-51. DDR SDRAM Single-Beat (Double Word) Write Timing—ACTTORW = 3**

### 8.5.4.1 Clock Distribution

- If running with many devices, zero-delay PLL clock buffers, JEDEC-JESD82 standard, should be used. These buffers were designed for DDR applications.
- A 72 bit x 64 Mbytes DDR bank has 9-byte-wide DDR chips, resulting in 18 DDR chips in a two-bank system. In this case, each MCK/$\overline{\text{MCK}}$ signal pair should drive exactly three devices.
- PCB traces for DDR clock signals should be short, all on the same layer, and of equal length and loading.
- DDR SDRAM manufacturers provide detailed information on PCB layout and termination issues.

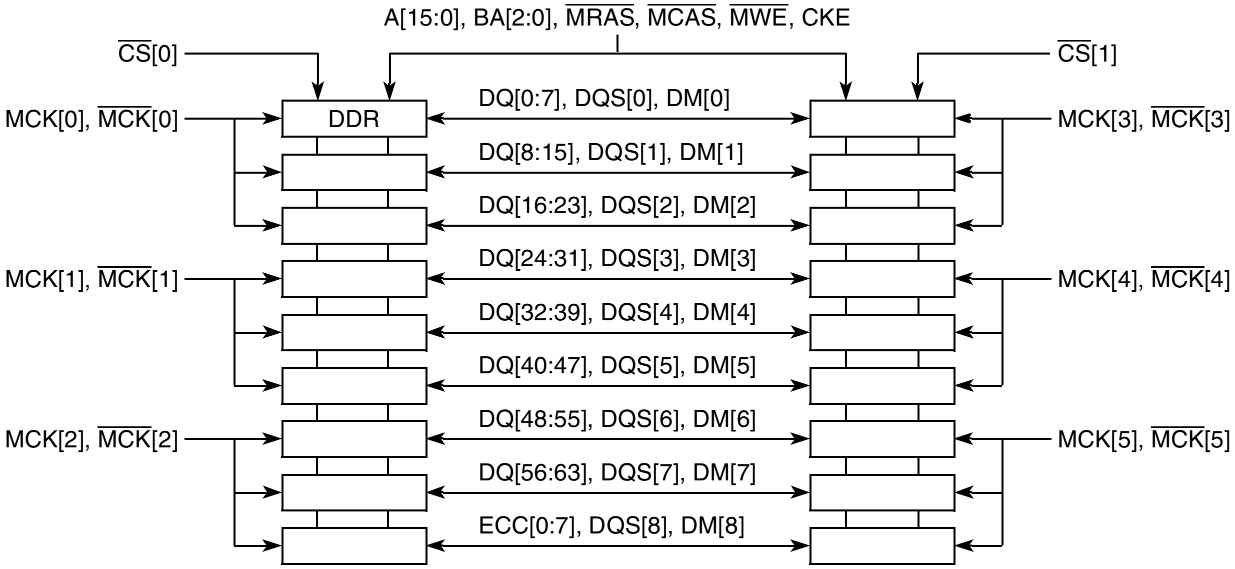

A[15:0], BA[2:0], $\overline{\text{MRAS}}$, $\overline{\text{MCAS}}$, $\overline{\text{MWE}}$, CKE
$\overline{\text{CS}}$[0]
$\overline{\text{CS}}$[1]
MCK[0], $\overline{\text{MCK}}$[0]
DDR
DQ[0:7], DQS[0], DM[0]
MCK[3], $\overline{\text{MCK}}$[3]
DQ[8:15], DQS[1], DM[1]
DQ[16:23], DQS[2], DM[2]
MCK[1], $\overline{\text{MCK}}$[1]
DQ[24:31], DQS[3], DM[3]
MCK[4], $\overline{\text{MCK}}$[4]
DQ[32:39], DQS[4], DM[4]
DQ[40:47], DQS[5], DM[5]
MCK[2], $\overline{\text{MCK}}$[2]
DQ[48:55], DQS[6], DM[6]
MCK[5], $\overline{\text{MCK}}$[5]
DQ[56:63], DQS[7], DM[7]
ECC[0:7], DQS[8], DM[8]

**Figure 8-52. DDR SDRAM Clock Distribution Example for x8 DDR SDRAMs**

## 8.5.5 DDR SDRAM Mode-Set Command Timing

The DDR memory controller transfers the mode register set commands to the SDRAM array, and it uses the setting of TIMING_CFG_0[MRS_CYC] for the Mode Register Set cycle time.

Figure 8-53 shows the timing of the mode-set command. The first transfer corresponds to the ESDMODE code; the second corresponds to SDMODE. The Mode Register Set cycle time is set to 2 DRAM cycles.

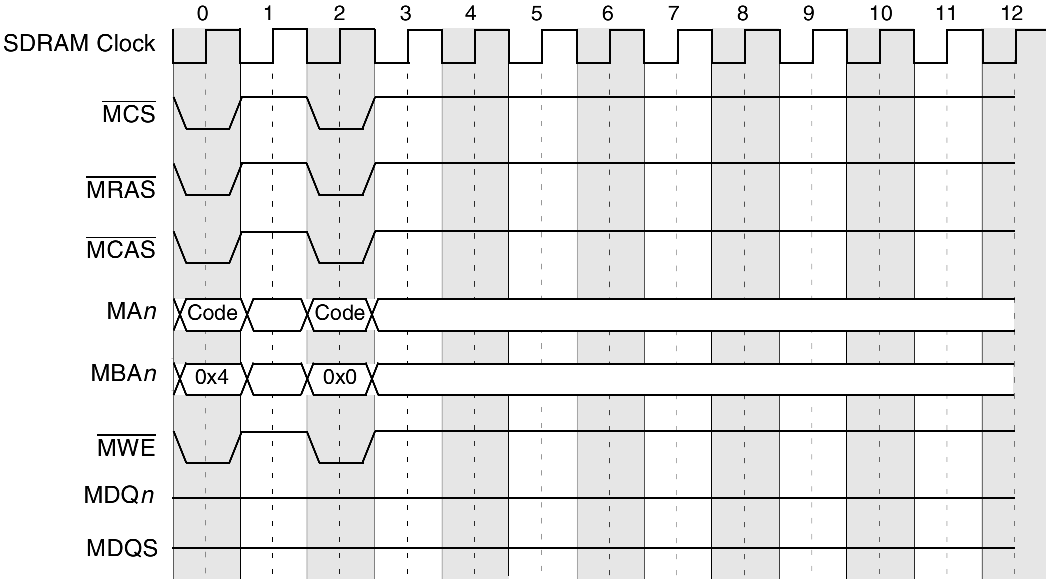


**Figure 8-53. DDR SDRAM Mode-Set Command Timing**

## 8.5.6 DDR SDRAM Registered DIMM Mode

To reduce loading, registered DIMMs latch the DDR SDRAM control signals internally before using them to access the array. Setting DDR_SDRAM_CFG[RD_EN] compensates for this delay on the DIMMs' control bus by delaying the data and data mask writes (on SDRAM buses) by an extra SDRAM clock cycle.

**NOTE**

Application system board must assert the reset signal on DDR memory devices until software is able to program the DDR memory controller configuration registers, and must deassert the reset signal on DDR memory devices before DDR_SDRAM_CFG[MEM_EN] is set. This ensures that the DDR memory devices are held in reset until a stable clock is provided and, further, that a stable clock is provided before memory devices are released from reset.

Figure 8-54 shows the registered DDR SDRAM DIMM single-beat write timing.

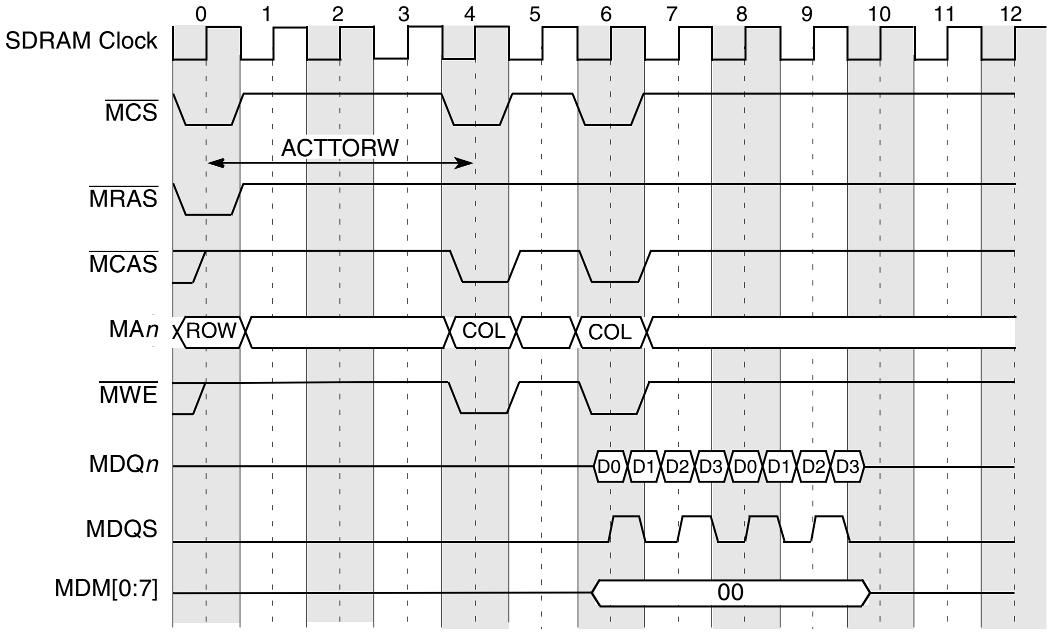


**Figure 8-54. Registered DDR SDRAM DIMM Burst Write Timing**

## 8.5.7 DDR SDRAM Write Timing Adjustments

The DDR memory controller facilitates system design flexibility by providing a write timing adjustment parameter, write data delay, (TIMING_CFG_2[WR_DATA_DELAY]) for data and DQS. The DDR SDRAM specification requires DQS be received no sooner than 75% of an SDRAM clock period—and no later than 125% of a clock period—from the capturing clock edge of the command/address at the SDRAM. The WR_DATA_DELAY parameter may be used to meet this timing requirement for a variety of system configurations, ranging from a system with one DIMM to a fully populated system with two DIMMs. TIMING_CFG_2[WR_DATA_DELAY] specifies how much to delay the launching of DQS and data from the first clock edge occurring one SDRAM clock cycle after the command is launched. The delay increment step sizes are in 1/4 SDRAM clock periods starting with the default value of 0.

Figure 8-55 shows the use of the WR_DATA_DELAY parameter.

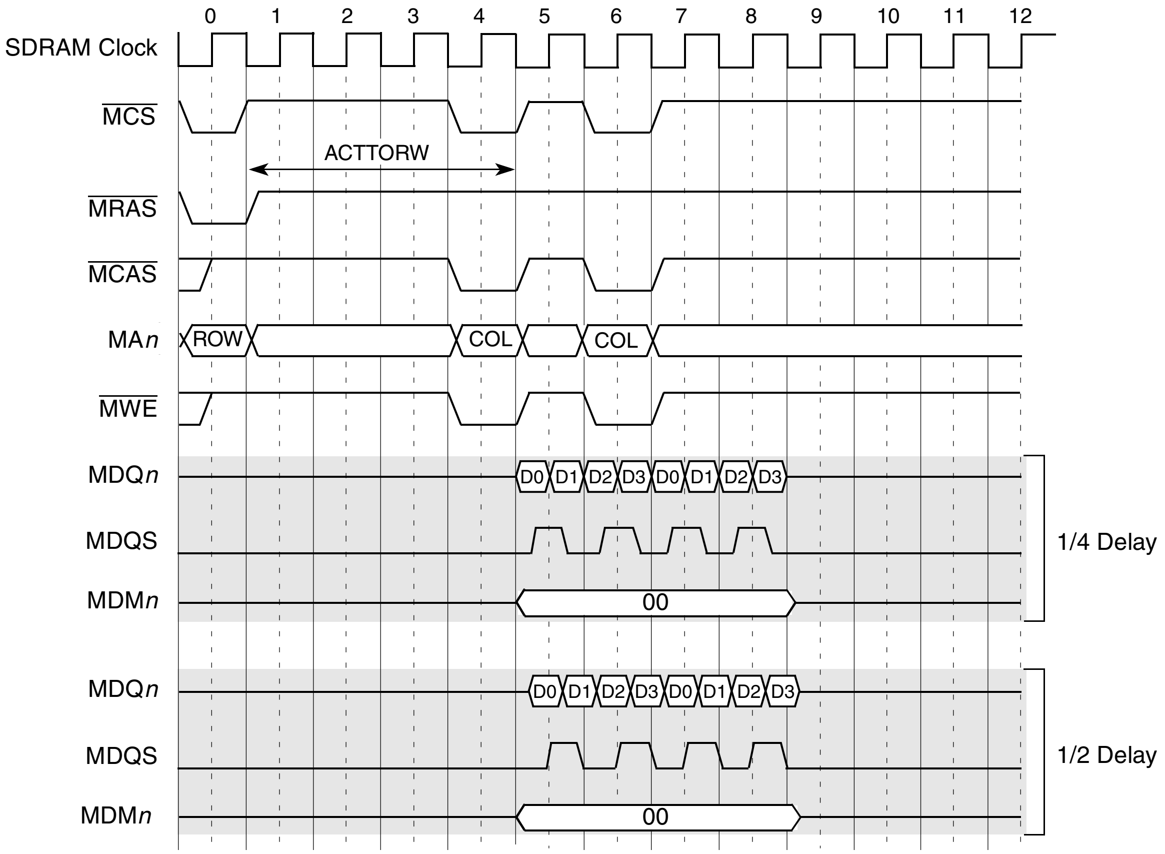


**Figure 8-55. Write Timing Adjustments Example for Write Latency = 1**

### 8.5.8 DDR SDRAM Refresh

The DDR memory controller supports auto-refresh and self-refresh. Auto refresh is used during normal operation and is controlled by the DDR_SDRAM_INTERVAL[REFINT] value; self-refresh is used only when the DDR memory controller is set to enter a sleep power management state. The REFINT value, which represents the number of memory bus clock cycles between refresh cycles, must allow for possible outstanding transactions to complete before a refresh request is sent to the memory after the REFINT value is reached. If a memory transaction is in progress when the refresh interval is reached, the refresh cycle waits for the transaction to complete. In the worst case, the refresh cycle must wait the number of bus clock cycles required by the longest programmed access. To ensure that the latency caused by a memory transaction does not violate the device refresh period, it is recommended that the programmed value of REFINT be less than that required by the SDRAM.

When a refresh cycle is required, the DDR memory controller does the following:

1. Completes all current memory requests.
2. Closes all open pages with a PRECHARGE-ALL command to each DDR SDRAM bank with an open page (as indicated by the row open table).
3. Issues one or more auto-refresh commands to each DDR SDRAM bank (as identified by its chip select) to refresh one row in each logical bank of the selected physical bank.

The auto-refresh commands are staggered across the four possible banks to reduce the system's instantaneous power requirements. Three sets of auto refresh commands are issued on consecutive cycles when the memory is fully populated with two DIMMs. The initial PRECHARGE-ALL commands are also staggered in three groups for convenience. It is important to note that when entering self-refresh mode, only one refresh command is issued simultaneously to all physical banks. For this entire refresh sequence, no cycle optimization occurs for the usual case where fewer than four banks are installed. After the refresh sequence completes, any pending memory request is initiated after an inactive period specified by TIMING_CFG_1 [REFREC] and TIMING_CFG_3[EXT_REFREC]. In addition, posted refreshes are supported to allow the refresh interval to be set to a larger value.

### 8.5.8.1 DDR SDRAM Refresh Timing

Refresh timing for the DDR SDRAM is controlled by the programmable timing parameter TIMING_CFG_1 [REFREC], which specifies the number of memory bus clock cycles from the refresh command until a logical bank activate command is allowed. The DDR memory controller implements bank staggering for refreshes, as shown in Figure 8-56 (TIMING_CFG_1 [REFREC] = 10 in this example).

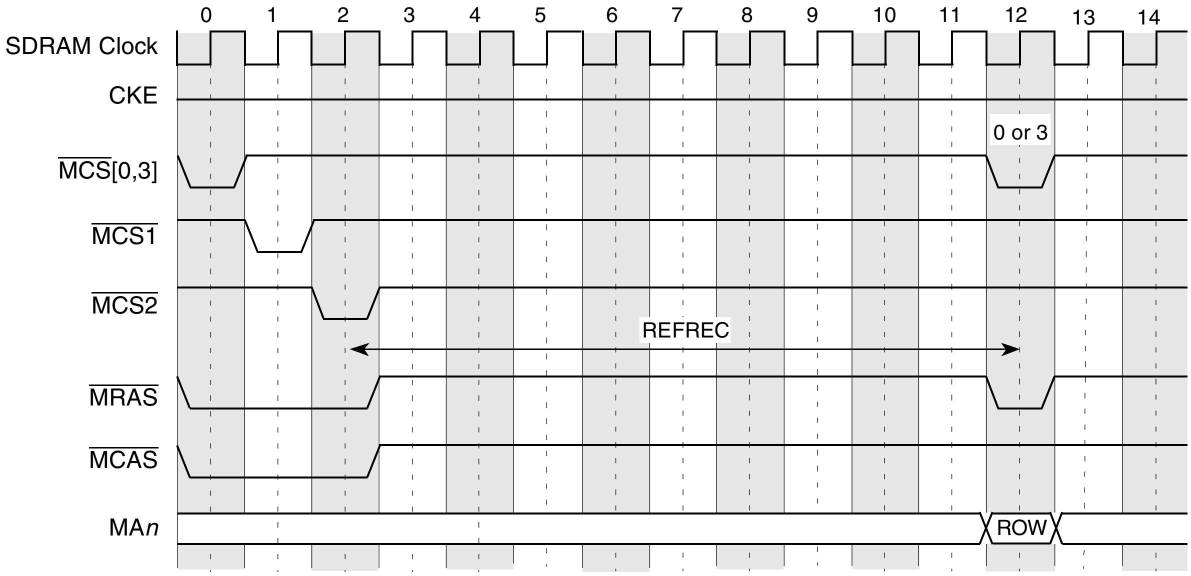


Figure 8-56. DDR SDRAM Bank Staggered Auto Refresh Timing

System software is responsible for optimal configuration of TIMING_CFG_1 [REFREC] and TIMING_CFG_3[EXT_REFREC] at reset. Configuration must be completed before DDR SDRAM accesses are attempted.

### 8.5.8.2 DDR SDRAM Refresh and Power-Saving Modes

In full-on mode, the DDR memory controller supplies the normal auto refresh to SDRAM. In sleep mode, the DDR memory controller can be configured to take advantage of self-refreshing SDRAMs or to provide no refresh support. Self-refresh support is enabled with the SREN memory control parameter.

Table 8-63 summarizes the refresh types available in each power-saving mode.

Table 8-63. DDR SDRAM Power-Saving Modes Refresh Configuration

| Power Saving Mode | Refresh Type | SREN |
|---|---|---|
| Sleep | Self | 1 |
|  | None | — |

Note that in the absence of refresh support, system software must preserve DDR SDRAM data (such as by copying the data to disk) before entering the power-saving mode.

The dynamic power-saving mode uses the CKE DDR SDRAM pin to dynamically power down when there is no system memory activity. The CKE pin is negated when both of the following conditions are met:

- No memory refreshes are scheduled
- No memory accesses are scheduled

CKE is reasserted when a new access or refresh is scheduled or the dynamic power mode is disabled. This mode is controlled with DDR_SDRAM_CFG[DYN_PWR_MGMT].

Dynamic power management mode offers tight control of the memory system's power consumption by trading power for performance through the use of CKE. Powering up the DDR SDRAM when a new memory reference is scheduled causes an access latency penalty, depending on whether active or precharge powerdown is used, along with the settings of TIMING_CFG_0[ACT_PD_EXIT] and TIMING_CFG_0[PRE_PD_EXIT]. A penalty of 1 cycle is shown in Figure 8-57.

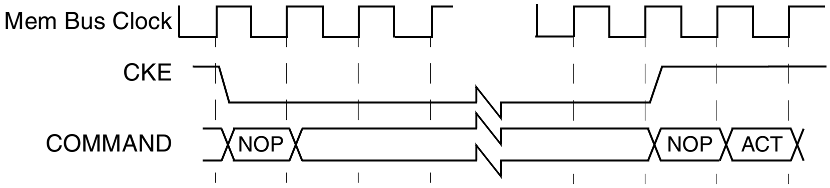


Figure 8-57. DDR SDRAM Power-Down Mode

### 8.5.8.2.1 Self-Refresh in Sleep Mode

The entry and exit timing for self-refreshing SDRAMs is shown in Figure 8-58 and Figure 8-59.

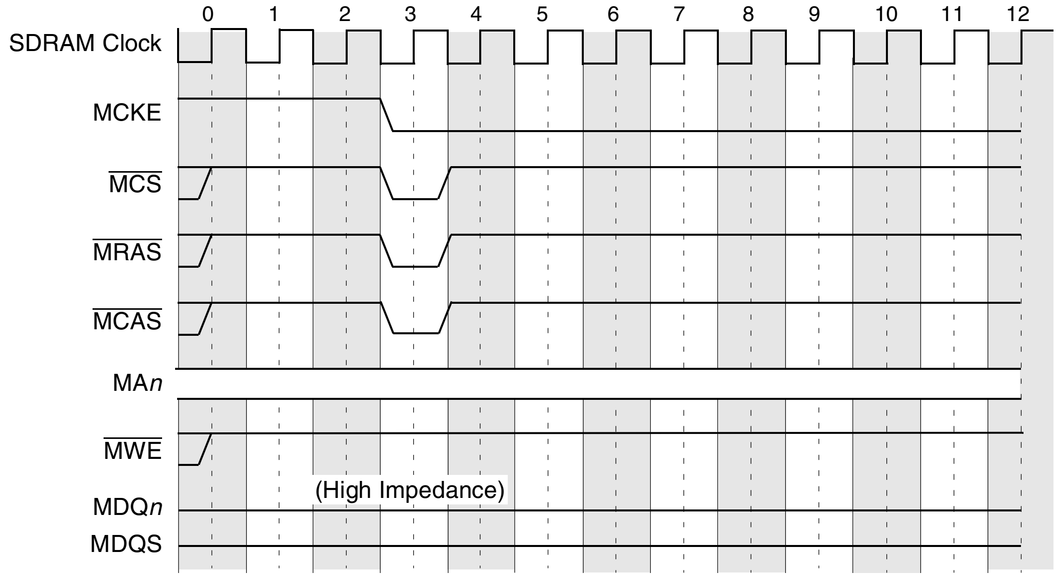


**Figure 8-58. DDR SDRAM Self-Refresh Entry Timing**

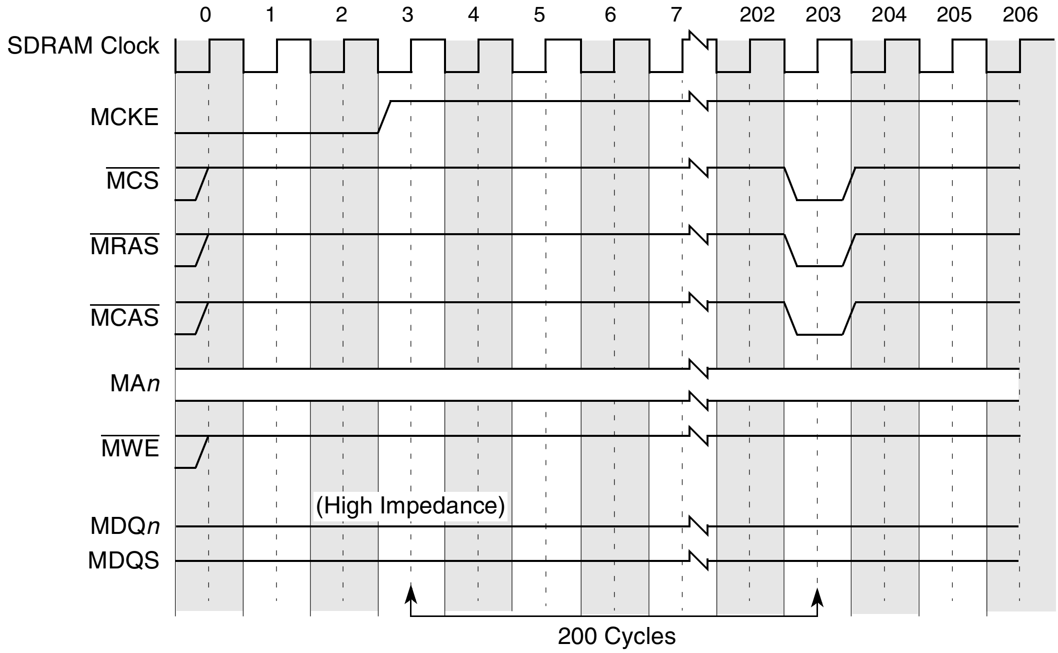


**Figure 8-59. DDR SDRAM Self-Refresh Exit Timing**

### 8.5.9 DDR Data Beat Ordering

Transfers to and from memory are always performed in four- or eight-beat bursts (four beats = 32 bytes when a 64-bit bus is used). For transfer sizes other than four or eight beats, the data transfers are still operated as four- or eight-beat bursts. If ECC is enabled and either the access is not doubleword aligned or the size is not a multiple of a doubleword, a full read-modify-write is performed for a write to SDRAM. If ECC is disabled or both the access is doubleword aligned with a size that is a multiple of a doubleword, the data masks (MDM[0:8] (MDM[0:4] for 32-bit bus) can be used to prevent the writing of unwanted data to SDRAM. The DDR memory controller also uses data masks to prevent all unintended full double words from writing to SDRAM. For example, if a write transaction is desired with a size of one double word (8 bytes), then the second, third, and fourth beats of data are not written to DRAM.

Table 8-64 lists the data beat sequencing to and from the DDR SDRAM and the data queues for each of the possible transfer sizes with each of the possible starting double-word offsets. All underlined double-word offsets are valid for the transaction.

**Table 8-64. Memory Controller–Data Beat Ordering**

| Transfer Size | Starting Double-Word Offset | Double-Word Sequence[1] to/from DRAM and Queues |
|---|---|---|
| 1 double word | 0<br>1<br>2<br>3 | **0** - 1 - 2 - 3<br>**1** - 2 - 3 - 0<br>**2** - 3 - 0 - 1<br>**3** - 0 - 1 - 2 |
| 2 double words | 0<br>1<br>2 | **0 - 1** - 2 - 3<br>**1 - 2** - 3 - 0<br>**2 - 3** - 0 - 1 |
| 3 double words | 0<br>1 | **0 - 1 - 2** - 3<br>**1 - 2 - 3** - 0 |

[1] All underlined **Double**-word offsets are valid for the transaction. All writes are aligned to double-word 0 for DDR3 memories.

### 8.5.10 Page Mode and Logical Bank Retention

The DDR memory controller supports an open/closed page mode with an allowable open page for each logical bank of DRAM used. In closed page mode for DDR SDRAMs, the DDR memory controller uses the SDRAM auto-precharge feature, which allows the controller to indicate that the page must be automatically closed by the DDR SDRAM after the READ or WRITE access. This is performed using MA[10] of the address during the COMMAND phase of the access to enable auto-precharge. Auto-precharge is non-persistent in that it is either enabled or disabled for each individual READ or WRITE command. It can, however, be enabled or disabled separately for each chip select.

When the DDR memory controller operates in open page mode, it retains the currently active SDRAM page by not issuing a precharge command. The page remains opens until one of the following conditions occurs:

- Refresh interval is met.
- The user-programmable DDR_SDRAM_INTERVAL[BSTOPRE] value is exceeded.

- There is a logical bank row collision with another transaction that must be issued.

Page mode can dramatically reduce access latencies for page hits. Depending on the memory system design and timing parameters, using page mode can save two to three clock cycles for subsequent burst accesses that hit in an active page. Also, better performance can be obtained using more banks, especially in systems which use many different channels. Page mode is disabled by clearing DDR_SDRAM_INTERVAL[BSTOPRE] or setting CS*n*_CONFIG[AP_*n*EN].

### 8.5.11 Error Checking and Correcting (ECC)

The DDR memory controller supports error checking and correcting (ECC) for the data path between the core master and system memory. The memory detects all double-bit errors, detects all multi-bit errors within a nibble, and corrects all single-bit errors. Other errors may be detected, but are not guaranteed to be corrected or detected. Multiple-bit errors are always reported when error reporting is enabled. When a single-bit error occurs, the single-bit error counter register is incremented, and its value compared to the single-bit error trigger register. An error is reported when these values are equal. The single-bit error registers can be programmed such that minor memory faults are corrected and ignored, but a catastrophic memory failure generates an interrupt.

For writes that are smaller than 64 bits, the DDR memory controller performs a double-word read from system memory of the address for the write (checking for errors), and merges the write data with the data read from memory. Then, a new ECC code is generated for the merged double word. The data and ECC code is then written to memory. If a multi-bit error is detected on the read, the transaction completes the read-modify-write to keep the DDR memory controller from hanging. However, the corrupt data is masked on the write, so the original contents in SDRAM remain unchanged.

The syndrome encodings for the ECC code are shown in Table 8-65 and Table 8-66.

In 32-bit mode, Table 8-65 is split into 2 halves. The first half, consisting of rows 0–31, is used to calculate the ECC bits for the first 32 data bits of any 64-bit granule of data. This always applies to the odd data beats on the DDR data bus. The second half of the table, consisting of rows 32–63, is used to calculate the ECC bits for the second 32 bits of any 64-bit granule of data. This always applies to the even data beats on the DDR data bus.

**Table 8-65. DDR SDRAM ECC Syndrome Encoding**

| Data Bit | Syndrome Bit | | | | | | | | Data Bit | Syndrome Bit | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0 | • | • | | | | | | • | 32 | | | • | • | | | | • |
| 1 | • | | • | | | | | • | 33 | | | • | | • | | | • |
| 2 | • | | | • | | | | • | 34 | • | | • | | • | | | |
| 3 | • | | | | • | | | • | 35 | | • | • | | • | | | |
| 4 | • | • | | | | • | | | 36 | | | • | • | | • | | |
| 5 | • | | • | | | • | | | 37 | | | • | | • | • | | |
| 6 | • | | | • | | • | | | 38 | • | | • | | • | • | | • |
| 7 | • | | | | • | • | | | 39 | | • | • | | • | • | | • |

Table 8-65. DDR SDRAM ECC Syndrome Encoding (continued)

| Data Bit | Syndrome Bit | | | | | | | | Data Bit | Syndrome Bit | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 8 | • | • | | | | | • | | 40 | | | • | • | | | • | |
| 9 | • | | • | | | | • | | 41 | | | • | | • | | • | |
| 10 | • | | | • | | | • | | 42 | • | | • | | • | | • | • |
| 11 | • | | | | • | | • | | 43 | | • | • | | • | | • | • |
| 12 | • | • | | | | • | • | • | 44 | | | • | • | | • | • | • |
| 13 | • | | • | | | • | • | • | 45 | | | • | | • | • | • | • |
| 14 | • | | | • | | • | • | • | 46 | • | | • | | • | • | • | |
| 15 | • | | | | • | • | • | • | 47 | | • | • | | • | • | • | |
| 16 | | • | • | | | | | • | 48 | | • | | | | • | • | |
| 17 | | • | | • | | | | • | 49 | | | • | | | • | • | |
| 18 | | • | | | • | | | • | 50 | | | | • | | • | • | |
| 19 | • | • | | | • | | | | 51 | • | | | | | • | • | |
| 20 | | • | • | | | • | | | 52 | | • | | | | • | | • |
| 21 | | • | | • | | • | | | 53 | | | • | | | • | | • |
| 22 | | • | | | • | • | | | 54 | | | | • | | • | | • |
| 23 | • | • | | | • | • | | • | 55 | • | | | | | • | | • |
| 24 | | • | • | | | | • | | 56 | | • | | | | | • | • |
| 25 | | • | | • | | | • | | 57 | | | • | | | | • | • |
| 26 | | • | | | • | | • | | 58 | | | | • | | | • | • |
| 27 | • | • | | | • | | • | • | 59 | • | | | | | | • | • |
| 28 | | • | • | | | • | • | • | 60 | | | | • | • | | • | |
| 29 | | • | | • | | • | • | • | 61 | • | | | • | • | | • | • |
| 30 | | • | | | • | • | • | • | 62 | | • | | • | • | | • | • |
| 31 | • | • | | | • | • | • | | 63 | | | • | • | • | | • | • |

Table 8-66. DDR SDRAM ECC Syndrome Encoding (Check Bits)

| Check Bit | Syndrome Bit | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0 | • | | | | | | | |
| 1 | | • | | | | | | |
| 2 | | | • | | | | | |
| 3 | | | | • | | | | |
| 4 | | | | | • | | | |
| 5 | | | | | | • | | |
| 6 | | | | | | | • | |
| 7 | | | | | | | | • |

## 8.5.12 Error Management

The DDR memory controller detects four different kinds of errors: training, single-bit, multi-bit, and memory select errors. The following discussion assumes all the relevant error detection, correction, and reporting functions are enabled as described in Section 8.4.1.39, “Memory Error Interrupt Enable (ERR_INT_EN),” Section 8.4.1.38, “Memory Error Disable (ERR_DISABLE),” and Section 8.4.1.37, “Memory Error Detect (ERR_DETECT).”

Single-bit errors are counted and reported based on the ERR_SBE value. When a single-bit error is detected, the DDR memory controller does the following:

- Corrects the data
- Increments the single-bit error counter ERR_SBE[SBEC]
- Generates a critical interrupt if the counter value ERR_SBE[SBEC] equals the programmable threshold ERR_SBE[SBET]
- Completes the transaction normally

If a multi-bit error is detected for a read, the DDR memory controller logs the error and generates the machine check or critical interrupt (if enabled, as described in Section 8.4.1.38, “Memory Error Disable (ERR_DISABLE)”). Another error the DDR memory controller detects is a memory select error, which causes the DDR memory controller to log the error and generate a critical interrupt (if enabled, as described in Section 8.4.1.37, “Memory Error Detect (ERR_DETECT)”). This error is detected if the address from the memory request does not fall into any of the enabled, programmed chip select address ranges. For all memory select errors, the DDR memory controller does not issue any transactions onto the pins after the first read has returned data strobes. If the DDR memory controller is not using sample points, then a dummy transaction is issued to DDR SDRAM with the first enabled chip select. In this case, the source port on the pins is forced to 0x1F to show the transaction is not real. Table 8-67 shows the errors with their descriptions. The final error the memory controller detects is the automatic calibration error. This error is set if the memory controller detects an error during its training sequence.

Table 8-67. Memory Controller Errors

| Category | Error | Descriptions | Action | Detect Register |
|---|---|---|---|---|
| Notification | Single-bit ECC threshold | The number of ECC errors has reached the threshold specified in the ERR_SBE. | The error is reported through machine check or critical interrupt if enabled. | The error control register only logs read versus write, not full type |
| Access Error | Multi-bit ECC error | A multi-bit ECC error is detected during a read, or read-modify-write memory operation. | | |
| | Memory select error | Read, or write, address does not fall within the address range of any of the memory banks. | | |

## 8.6 Initialization/Application Information

System software must configure the DDR memory controller, using a memory polling algorithm at system start-up, to correctly map the size of each bank in memory. Then, the DDR memory controller uses its bank map to assert the appropriate $\overline{\mathrm{MCS}n}$ signal for memory accesses according to the provided bank depths. System software must also configure the DDR memory controller at system start-up to appropriately multiplex the row and column address bits for each bank. Refer to row-address configuration in Section 8.4.1.2, “Chip Select Configuration (CSn_CONFIG).” Address multiplexing occurs according to these configuration bits.

At system reset, initialization software (boot code) must set up the programmable parameters in the memory interface configuration registers. See Section 8.4.1, “Register Descriptions,” for more detailed descriptions of the configuration registers. These parameters are shown in Table 8-68.

Table 8-68. Memory Interface Configuration Register Initialization Parameters

| Name | Description | Parameter | Section/page |
|---|---|---|---|
| CS*n*_BNDS | Chip select memory bounds | SA*n*<br>EA*n* | 8.4.1.1/8-12 |
| CS*n*_CONFIG | Chip select configuration | CS_*n*_EN<br>AP_*n*_EN<br>ODT_RD_CFG<br>ODT_WR_CFG<br>BA_BITS_CS_*n*<br>ROW_BITS_CS_*n*<br>COL_BITS_CS_*n* | 8.4.1.2/8-13 |
| CS*n*_CONFIG_2 | Chip select configuration 2 | PASR_CFG | 8.4.1.3/8-15 |
| TIMING_CFG_3 | Extended timing parameters for fields in TIMING_CFG_1 | EXT_REFREC<br>EXT_ACTTOPRE<br>EXT_CASLAT<br>CNTL_ADJ | 8.4.1.4/8-16 |
| TIMING_CFG_0 | Timing configuration | RWT<br>WRT<br>RRT<br>WWT<br>ACT_PD_EXIT<br>PRE_PD_EXIT<br>ODT_PD_EXIT<br>MRS_CYC | 8.4.1.5/8-17 |
| TIMING_CFG_1 | Timing configuration | PRETOACT<br>ACTTOPRE<br>ACTTORW<br>CASLAT<br>REFREC<br>WRREC<br>ACTTOACT<br>WRTORD | 8.4.1.6/8-19 |

**Table 8-68. Memory Interface Configuration Register Initialization Parameters (continued)**

| Name | Description | Parameter | Section/page |
|---|---|---|---|
| TIMING_CFG_2 | Timing configuration | ADD_LAT<br>CPO<br>WR_LAT<br>RD_TO_PRE<br>WR_DATA_DELAY<br>CKE_PLS<br>FOUR_ACT | 8.4.1.7/8-21 |
| DDR_SDRAM_CFG | Control configuration | SREN<br>ECC_EN<br>RD_EN<br>SDRAM_TYPE<br>DYN_PWR<br>32_BE<br>8_BE<br>DBW<br>NCAP<br>2T_EN<br>3T_EN<br>BA_INTLV_CTL<br>x32_EN<br>HSE<br>BI | 8.4.1.8/8-23 |
| DDR_SDRAM_CFG_2 | Control configuration | SR_IE<br>DLL_RST_DIS<br>DQS_CFG<br>ODT_CFG<br>NUM_PR<br>OBC_CFG<br>AP_EN<br>D_INIT<br>RCW_EN<br>MD_EN | 8.4.1.9/8-26 |
| DDR_SDRAM_MODE | Mode configuration | ESDMODE<br>SDMODE | 8.4.1.10/8-29 |
| DDR_SDRAM_MODE_2 | Mode configuration | ESDMODE2<br>ESDMODE3 | 8.4.1.11/8-29 |
| DDR_SDRAM_INTERVAL | Interval configuration | REFINT<br>BSTOPRE | 8.4.1.13/8-33 |
| DDR_DATA_INIT | Data initialization configuration register | INIT_VALUE | 8.4.1.14/8-33 |
| DDR_SDRAM_CLK_CNTL | Clock adjust | CLK_ADJUST | 8.4.1.15/8-34 |
| DDR_INIT_ADDR | Initialization address | INIT_ADDR | 8.4.1.16/8-34 |
| TIMING_CFG_4 | Timing configuration | RWT<br>WRT<br>RRT<br>WWT<br>DLL_LOCK | 8.4.1.18/8-36 |
| TIMING_CFG_5 | Timing configuration | RODT_ON<br>RODT_OFF<br>WODT_ON<br>WODT_OFF | 8.4.1.19/8-37 |
| DDR_ZQ_CNTL | ZQ calibration control | ZQ_EN<br>ZQINIT<br>ZQOPER<br>ZQCS | 8.4.1.20/8-39 |

**Table 8-68. Memory Interface Configuration Register Initialization Parameters (continued)**

| Name | Description | Parameter | Section/page |
|---|---|---|---|
| DDR_WRLVL_CNTL | Write leveling control | WRLVL_EN<br>WRLVL_MRD<br>WRLVL_ODTEN<br>WRLVL_DQSEN<br>WRLVL_SMPL<br>WRLVL_WLR<br>WRLVL_START | 8.4.1.21/8-40 |
| DDR_SR_CNTR | Self refresh control | SR_IT | 8.4.1.22/8-43 |
| DDR_SDRAM_RCW_1 | Register control words configuration | RCW0<br>RCW1<br>RCW2<br>RCW3<br>RCW4<br>RCW5<br>RCW6<br>RCW7 | 8.4.1.23/8-44 |
| DDR_SDRAM_RCW_2 | Register control words configuration | RCW8<br>RCW9<br>RCW10<br>RCW11<br>RCW12<br>RCW13<br>RCW14<br>RCW15 | 8.4.1.24/8-45 |
| DDRCDR_1 | Driver control | DHC_EN<br>ODT<br>DSO_C_EN<br>DSO_D_EN<br>DSO_CPZ<br>DSO_CNZ<br>DSO_DPZ<br>DSO_DNZ | 8.4.1.27/8-47 |
| DDRCDR_2 | Driver control | DSO_CLK_EN<br>DSO_CLKPZ<br>DSO_CLKNZ | 8.4.1.28/8-50 |

## 8.6.1 Programming Differences between Memory Types

Depending on the memory type used, certain fields must be programmed differently. Table 8-69 illustrates the differences in certain fields for different memory types. Note: This table does not list all fields that must be programmed.

**Table 8-69. Programming Differences Between Memory Types**

| Parameter | Description | Differences | | Section/page |
|---|---|---|---|---|
| AP*n*_EN | Chip Select *n* Auto Precharge Enable | DDR2 | Can be used to place chip select *n* in auto precharge mode | 8.4.1.2/8-13 |
| | | DDR3 | Can be used to place chip select *n* in auto precharge mode | |

Table 8-69. Programming Differences Between Memory Types (continued)

| Parameter | Description | Differences | | Section/page |
|---|---|---|---|---|
| ODT_RD_CFG | Chip Select ODT Read Configuration | DDR2 | Can be enabled to assert ODT if desired. This could be set differently depending on system topology. However, systems with only 1 chip select will typically not use ODT when issuing reads to the memory. | 8.4.1.2/8-13 |
| | | DDR3 | Can be enabled to assert ODT if desired. This could be set differently depending on system topology. However, systems with only 1 chip select typically do not use ODT when issuing reads to the memory. | |
| ODT_WR_CFG | Chip Select ODT Write Configuration | DDR2 | Can be enabled to assert ODT if desired. This could be set differently depending on system topology. However, ODT will typically be set to assert for the chip select that is getting written to (value would be set to 001). | 8.4.1.2/8-13 |
| | | DDR3 | Can be enabled to assert ODT if desired. This could be set differently depending on system topology. However, ODT typically is set to assert for the chip select that is getting written to (value would be set to 001). | |
| ODT_PD_EXIT | ODT Powerdown Exit | DDR2 | Should be set according to the DDR2 specifications for the memory used. The JEDEC parameter this applies to is $t_{AXPD}$. | 8.4.1.5/8-17 |
| | | DDR3 | Should be set to 0001 for DDR3. The powerdown times ($t_{XP}$ and $t_{XPDLL}$) required for DDR3 are controlled via TIMING_CFG_0[ACT_PD_EXIT] and TIMING_CFG_0[PRE_PD_EXIT]. | |
| PRETOACT | Precharge to Activate Timing | DDR2 | Should be set according to the specifications for the memory used ($t_{RP}$) | 8.4.1.6/8-19 |
| | | DDR3 | Should be set according to the specifications for the memory used ($t_{RP}$) | |
| ACTTOPRE | Activate to Precharge Timing | DDR2 | Should be set, along with the Extended Activate to Precharge Timing, according to the specifications for the memory used ($t_{RAS}$) | 8.4.1.6/8-19 |
| | | DDR3 | Should be set, along with the Extended Activate to Precharge Timing, according to the specifications for the memory used ($t_{RAS}$) | |
| ACTTORW | Activate to Read/Write Timing | DDR2 | Should be set according to the specifications for the memory used ($t_{RCD}$) | 8.4.1.6/8-19 |
| | | DDR3 | Should be set according to the specifications for the memory used ($t_{RCD}$) | |
| CASLAT | CAS Latency | DDR2 | Should be set, along with the Extended CAS Latency, to the desired CAS latency | 8.4.1.6/8-19 |
| | | DDR3 | Should be set, along with the Extended CAS Latency, to the desired CAS latency | |

Table 8-69. Programming Differences Between Memory Types (continued)

| Parameter | Description | Differences | | Section/page |
|---|---|---|---|---|
| REFREC | Refresh Recovery | DDR2 | Should be set, along with the Extended Refresh Recovery, to the specifications for the memory used ($T_{RFC}$) | 8.4.1.6/8-19 |
| | | DDR3 | Should be set, along with the Extended Refresh Recovery, to the specifications for the memory used ($T_{RFC}$) | |
| WRREC | Write Recovery | DDR2 | Should be set according to the specifications for the memory used ($t_{WR}$) | 8.4.1.6/8-19 |
| | | DDR3 | Should be set according to the specifications for the memory used ($t_{WR}$). If DDR_SDRAM_CFG_2[OBC_CFG] is set, then this should be programmed to $t_{WR}$ + 2 DRAM cycles. | |
| ACTTOACT | Activate *A* to Activate *B* | DDR2 | Should be set according to the specifications for the memory used ($t_{RRD}$) | 8.4.1.6/8-19 |
| | | DDR3 | Should be set according to the specifications for the memory used ($t_{RRD)}$ | |
| WRTORD | Write to Read Timing | DDR2 | Should be set according to the specifications for the memory used ($t_{WTR}$) | 8.4.1.6/8-19 |
| | | DDR3 | Should be set according to the specifications for the memory used ($t_{WTR}$)<br>If DDR_SDRAM_CFG_2[OBC_CFG] is set, then this should be programmed to $t_{WTR}$ + 2 DRAM cycles. | |
| ADD_LAT | Additive Latency | DDR2 | Should be set to the desired additive latency. This must be set to a value less than TIMING_CFG_1[ACTTORW] | 8.4.1.7/8-21 |
| | | DDR3 | Should be set to the desired additive latency. This must be set to a value less than TIMING_CFG_1[ACTTORW] | |
| WR_LAT | Write Latency | DDR2 | Should be set to CAS latency – 1 cycle. For example, if the CAS latency if 5 cycles, then this field should be set to 100 (4 cycles). | 8.4.1.7/8-21 |
| | | DDR3 | Should be set to the desired write latency. Note that DDR3 SDRAMs do not necessarily require the write latency to equal the CAS latency minus 1 cycle. | |

### Table 8-69. Programming Differences Between Memory Types (continued)

| Parameter | Description | Differences | | Section/page |
|---|---|---|---|---|
| RD_TO_PRE | Read to Precharge Timing | DDR2 | Should be set according to the specifications for the memory used ($t_{RTP}$). Time between read and precharge for non-zero value of additive latency (AL) is a minimum of AL + $t_{RTP}$ cycles. | 8.4.1.7/8-21 |
| | | DDR3 | Should be set according to the specifications for the memory used ($t_{RTP}$). Time between read and precharge for non-zero value of additive latency (AL) is a minimum of AL + $t_{RTP}$ cycles. If DDR_SDRAM_CFG_2[OBC_CFG] is set, then this should be programmed to $t_{RTP}$ + 2 DRAM cycles. | |
| CKE_PLS | Minimum CKE Pulse Width | DDR2 | Should be set according to the specifications for the memory used ($t_{CKE}$) | 8.4.1.7/8-21 |
| | | DDR3 | Should be set according to the specifications for the memory used ($t_{CKE}$) | |
| FOUR_ACT | Four Activate Window | DDR2 | Should be set according to the specifications for the memory used ($t_{FAW}$). Only applies to eight logical banks. | 8.4.1.7/8-21 |
| | | DDR3 | Should be set according to the specifications for the memory used ($t_{FAW}$). | |
| RD_EN | Registered DIMM Enable | DDR2 | If registered DIMMs are used, then this field should be set to 1 | 8.4.1.8/8-23 |
| | | DDR3 | If registered DIMMs are used, then this field should be set to 1 | |
| 8_BE | 8-beat burst enable | DDR2 | Should be set to 0 | 8.4.1.8/8-23 |
| | | DDR3 | If a 64-bit bus is used, this should be set to 0. Otherwise, this should be set to 1. If this is set to 0, then other requirements in TIMING_CFG_4 is needed to ensure $t_{CCD}$ is met. | |
| 2T_EN | 2T Timing Enable | DDR2 | In heavily loaded systems, this can be set to 1 to gain extra timing margin on the interface at the cost of address/command bandwidth. | 8.4.1.8/8-23 |
| | | DDR3 | In heavily loaded systems, this can be set to 1 to gain extra timing margin on the interface at the cost of address/command bandwidth. | |
| DLL_RST_DIS | DLL Reset Disable | DDR2 | Should typically be set to 0, unless it is desired to bypass the DLL reset when exiting self refresh. | 8.4.1.9/8-26 |
| | | DDR3 | Should be set to 1 | |
| DQS_CFG | DQS Configuration | DDR2 | Should be set to 01 | 8.4.1.9/8-26 |
| | | DDR3 | Should be set to 01 | |

### Table 8-69. Programming Differences Between Memory Types (continued)

| Parameter | Description | Differences | | Section/page |
|---|---|---|---|---|
| ODT_CFG | ODT Configuration | DDR2 | Can be set for termination at the IOs according to system topology. Typically, if ODT is enabled, then the internal IOs should be set up for termination only during reads to DRAM. | 8.4.1.9/8-26 |
| | | DDR3 | Can be set for termination at the IOs according to system topology. Typically, if ODT is enabled, then the internal IOs should be set up for termination only during reads to DRAM. | |
| OBC_CFG | On-The-Fly Burst Chop Configuration | DDR2 | Should be set to 0 | 8.4.1.9/8-26 |
| | | DDR3 | Can be set to 1 if on-the-fly burst chop is used. This is expected to give the best performance in DDR3 mode. This feature can only be used if a 64-bit data bus is used. | |
| RWT | Read-to-write turnaround for same chip select (in TIMING_CFG_4) | DDR2 | Should typically be set to 0000 | 8.4.1.18/8-36 |
| | | DDR3 | This can be used to force a longer read-to-write turnaround time when accessing the same chip select. This is useful for burst chop mode, as there are some timing requirements to the same chip select that still must be met. | |
| WRT | Write-to-read turnaround for same chip select (in TIMING_CFG_4) | DDR2 | Should typically be set to 0000 | 8.4.1.18/8-36 |
| | | DDR3 | This could be used to force a certain turnaround time between a write and read to the same chip select. This is useful for burst chop mode. However, it is expected that TIMING_CFG_1[WRTORD] is programmed appropriately such that TIMING_CFG_4[WRT] can be set to 0000. | |
| RRT | Read-to-read turnaround for same chip select (in TIMING_CFG_4) | DDR2 | Should typically be set to 0000 | 8.4.1.18/8-36 |
| | | DDR3 | Should typically be set to 0100 in burst chop mode (on-the-fly or fixed). | |
| WWT | Write-to-write turnaround for same chip select (in TIMING_CFG_4) | DDR2 | Should typically be set to 0000 | 8.4.1.18/8-36 |
| | | DDR3 | Should typically be set to 0100 in burst chop mode (on-the-fly or fixed). | |
| ZQ_EN | ZQ Calibration Enable | DDR2 | Should be set to 0 | 8.4.1.20/8-39 |
| | | DDR3 | Should be set to 1. The other fields in DDR_ZQ_CNTL should also be programmed appropriately based on the DRAM specifications. | |

Table 8-69. Programming Differences Between Memory Types (continued)

| Parameter | Description | Differences | | Section/page |
|---|---|---|---|---|
| WRLVL_EN | Write Leveling Enable | DDR2 | Should be set to 0 | 8.4.1.21/8-40 |
| | | DDR3 | Can be set to 1 if write leveling is desired. Otherwise the value used in TIMING_CFG_2[WR_DATA_DELAY] is used to shift all bytes during writes to DRAM. If write leveling is used, all other fields in DDR_WRLVL_CNTL should be programmed appropriately based on the DRAM specifications. | |
| BSTOPR | Burst To Precharge Interval | DDR2 | Can be set to any value, depending on the application. Auto precharge can be enabled by setting this field to all 0s. | 8.4.1.13/8-33 |
| | | DDR3 | Can be set to any value, depending on the application. Auto precharge can be enabled by setting this field to all 0s. | |

## 8.6.2 DDR SDRAM Initialization Sequence

After configuration of all parameters is complete, system software must set DDR_SDRAM_CFG[MEM_EN] to enable the memory interface. Note that 200 µs (500 µs for DDR3) must elapse after DRAM clocks are stable (DDR_SDRAM_CLK_CNTL[CLK_ADJUST] is set and any chip select is enabled) before MEM_EN can be set, so a delay loop in the initialization code may be necessary if software is enabling the memory controller. If DDR_SDRAM_CFG[BI] is not set, the DDR memory controller conducts an automatic initialization sequence to the memory, which follows the memory specifications. If the bypass initialization mode is used, then software can initialize the memory through the DDR_SDRAM_MD_CNTL register.

## 8.6.3 Using Forced Self-Refresh Mode to Implement a Battery-Backed RAM System

This section describes the options offered by this device to support battery-backed main memory.

### 8.6.3.1 Hardware Based Self-Refresh

An external voltage sense device can be connected to this device through one of the external interrupt lines IRQ*n*. The external interrupt from the voltage sensor would then be steered through this device’s programmable interrupt controller (PIC) to the $\overline{\text{IRQ\_OUT}}$ signal. Note that the $\overline{\text{IRQ\_OUT}}$ signal must remain high until power is removed.

If DDR_SDRAM_CFG_2[SR_IE] is set, the $\overline{\text{IRQ\_OUT}}$ signal from the interrupt controller is then automatically detected by the DDR controller, which immediately causes main memory to enter self-refresh mode. See Section 8.4.1.9, “DDR SDRAM Control Configuration 2 (DDR_SDRAM_CFG_2),” for further information on this bit.

These fields in the appropriate registers in the PIC must be set for self refresh to function:

- EIVPR*n*[PRIORITY] should be set to 0xF (highest priority)
- EIDR*n*[EP] should be set in order to route the incoming signal to $\overline{\text{IRQ\_OUT}}$

See Section 9.3.7.1, "External Interrupt Vector/Priority Registers (EIVPR0–EIVPR11)," and Section 9.3.7.2, "External Interrupt Destination Registers (EIDR0–EIDR11)," for descriptions of these registers.

Note that this application precludes any other simultaneous use of $\overline{\text{IRQ\_OUT}}$.

### 8.6.3.2 Software Based Self-Refresh

The DDR controller also has a software-programmable bit, DDR_SDRAM_CFG_2[FRC_SR], that immediately puts main memory into self-refresh mode. See Section 8.4.1.9, "DDR SDRAM Control Configuration 2 (DDR_SDRAM_CFG_2)," for a description of this register.

It is expected that a critical interrupt routine triggered by an external voltage sensing device has time to set this bit.

### 8.6.3.3 Bypassing Re-initialization During Battery-Backed Operation

The DDR controller offers an initialization bypass feature (DDR_SDRAM_CFG[BI]), which system designers may use to prevent re-initialization of main memory during system power-on following an abnormal shutdown. See Section 8.4.1.8, "DDR SDRAM Control Configuration (DDR_SDRAM_CFG)," for information on this bit and Section 8.4.1.16, "DDR Initialization Address (DDR_INIT_ADDR)," for a discussion of avoiding possible ECC errors in this mode.

Note that the DDR controller automatically waits 200 DRAM cycles before issuing any command after the assertion of MCKE[0:3] when this mode is used.

# Chapter 9
# Programmable Interrupt Controller (PIC)

This chapter describes the programmable interrupt controller (PIC) interrupt protocol, various types of interrupt sources controlled by the PIC, and the PIC registers with some programming guidelines.

## 9.1 Introduction

The PIC conforms to the OpenPIC architecture. The interrupt controller provides multiprocessor interrupt management, and is responsible for receiving hardware-generated interrupts from different sources (both internal and external), prioritizing them, and delivering them to a CPU for servicing.

### 9.1.1 Overview

Figure 9-1 is a block diagram showing the relationship of the various functional blocks and how the signals external to the PIC are connected to other blocks on the device, including the cores.

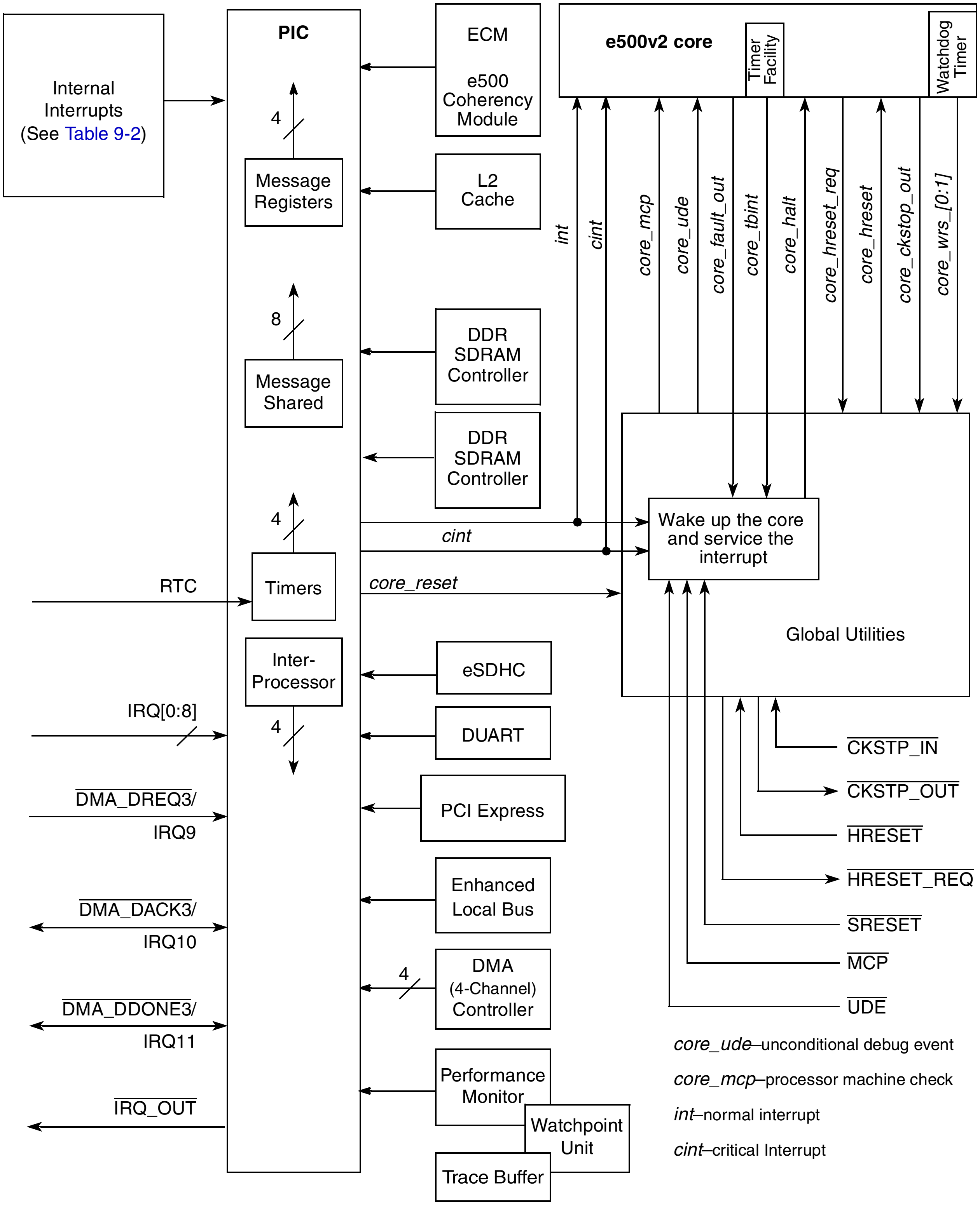


**Figure 9-1. Interrupt Sources Block Diagram Features**

The PIC has the following features:

- Support for the following interrupt sources:
  - External—Off-chip signals, IRQ[0:11]
  - Internal—These are on-chip sources from peripheral logic within the integrated device signalling error conditions that need to be addressed by software.
- Interrupts generated from within the PIC itself, which are as follows:
  - Global timers A and B internal to the PIC
  - Interprocessor interrupts (IPI)—Intended for communication between different processor cores on the same device. (Can be used for self-interrupt in single-core devices.)
  - Message registers—From within the PIC. Triggered on register write, cleared on read. Used for interprocessor communication.
  - Shared message signaled registers—From within the PIC. Triggered on register write, cleared on read. Used for cross-program communication.
    - Eight 32-bit message interrupt channels.
    - Two groups of four global 32-bit timers clocked with the CCB clock or the RTC input. Timers within each group can be concatenated to time longer durations.
- Three types of programmable interrupt outputs:
  - External interrupt (*int*). Any of the PIC interrupt sources can be programmed to direct interrupt requests to *int*. Handling of such interrupt requests follows the OpenPIC specification, which guarantees that the highest priority interrupt supersedes lower priority interrupts. Section 9.4.1.2, “Interrupts Routed to *int*,” describes how the PIC logic handles these interrupts.
  - Critical interrupt (*cint*). Connected to the respective core’s critical interrupt input.
  - $\overline{\text{IRQ\_OUT}}$.

    Section 9.4.1.1, “Interrupts Routed to *cint* or IRQ_OUT,” describes how the PIC logic supports this interrupt.
- Programming model compliant with the OpenPIC architecture.
  - Message, interprocessor and global timer interrupts. (Note that the interprocessor and global timer interrupts can only be routed to *int*.)
  - The following OpenPIC-defined features support only interrupts routed to the *int* signal:
    - Fully-nested interrupt delivery, guaranteeing that the interrupt source with the highest priority is given precedence over lower priority interrupts, including any that are in service.
    - 16 programmable interrupt priority levels
    - Support for identifying and handling spurious interrupts
- Support for two processors.
  - Interrupts can be routed to processor core 0 or 1
  - Multi-cast delivery mode for interprocessor and global timer interrupts allowing these interrupts to be routed to either core 0 or 1, or both cores.
- Processor core initialization control
- Programmable resetting of the PIC through the global configuration register

- Support for connection of external interrupt controller device such as an 8259 programmable interrupt controller. In 8259 mode, an interrupt causes assertion of a local (that is, internal to the integrated device) interrupt output signal.
- Pass-through mode (PIC disabled) in which the PIC directs interrupts off-chip for external servicing. See Section 9.1.3.2, "Pass-Through Mode (GCR[M] = 0)."

## 9.1.2 Interrupts to the Processor Core

The external interrupt signal, *int*, is the main interrupt output from the PIC to the processor core.

The interrupt sources can also specify the critical interrupt output, c*int*, if the corresponding *x*IDR*n*[CI0] or *x*IDR*n*[CI1] is set.

The PIC also defines the PIR, described in Section 9.3.1.6, "Processor Core Initialization Register (PIR)," which can be used to reset the core. Processor core interrupts generated by the PIC are described in Table 9-1.

**Table 9-1. Processor Interrupts Generated Outside the Core—Types and Sources**

| Core Interrupt Type | Signaled by (Input to Core) | Sources |
|---|---|---|
| **PIC-Programmable Interrupts** | | |
| External interrupt | *int* | Generated by the PIC, as described in Section 9.1.4, "Interrupt Sources." |
| Critical interrupt | *cint* | Generated by the PIC, as described in Section 9.1.4, "Interrupt Sources." |
| **Other Interrupts Generated Outside the Core** | | |
| Machine check | *coren_mcp* | • $\overline{\text{MCP}}$<br>• $\overline{\text{SRESET}}$<br>• Assertion of *core_mcp* by global utilities block |
| Unconditional debug event | *coren_ude* | $\overline{\text{UDE}}$. Asserting $\overline{\text{UDE}}$ generates an unconditional debug exception type debug interrupt and sets a bit in the debug status register, DBSR[UDE], as described in Section 6.13.2, "Debug Status Register (DBSR)." |
| Reset | *coren_hreset* | • $\overline{\text{HRESET}}$ assertion (and negation)<br>• *core_hreset_req*. Output from core—caused by writing to the core DBCR0[RST]. This condition is additionally qualified with MSR[DE] and DBCR0[IDM] bits. Note that assertion of this signal causes a hard reset of the core only. *core_hreset_req* can also be caused by a second timer timeout condition as described in Section 9.3.2.6, "Timer Control Registers (TCRA–TCRB)."<br>• *core_reset*. Output from PIC. See Section 9.3.1.6, "Processor Core Initialization Register (PIR)." |

## 9.1.3 Modes of Operation

Mixed or pass-through mode of operation is chosen by setting or clearing GCR[M], as described in Section 9.3.1.4, "Global Configuration Register (GCR)."

### 9.1.3.1 Mixed Mode (GCR[M] = 1)

In mixed mode, external and internal interrupts are delivered using the normal priority and delivery mechanisms detailed in Section 9.4.1, “Flow of Interrupt Control.”

### 9.1.3.2 Pass-Through Mode (GCR[M] = 0)

The PIC provides a mechanism to support alternate external interrupt controllers such as the PC/AT-compatible 8259 interrupt controller architecture. After a hard reset, the PIC defaults to pass-through mode, in which active-high interrupts from external source IRQ0 are passed directly to core 0 as shown in Figure 9-2; all other external interrupt signals are ignored. Thus, the interrupt signal from an external interrupt controller can be connected to IRQ0 and cause direct interrupts to the processor core 0. The PIC does not perform a vector fetch from an 8259 interrupt controller.

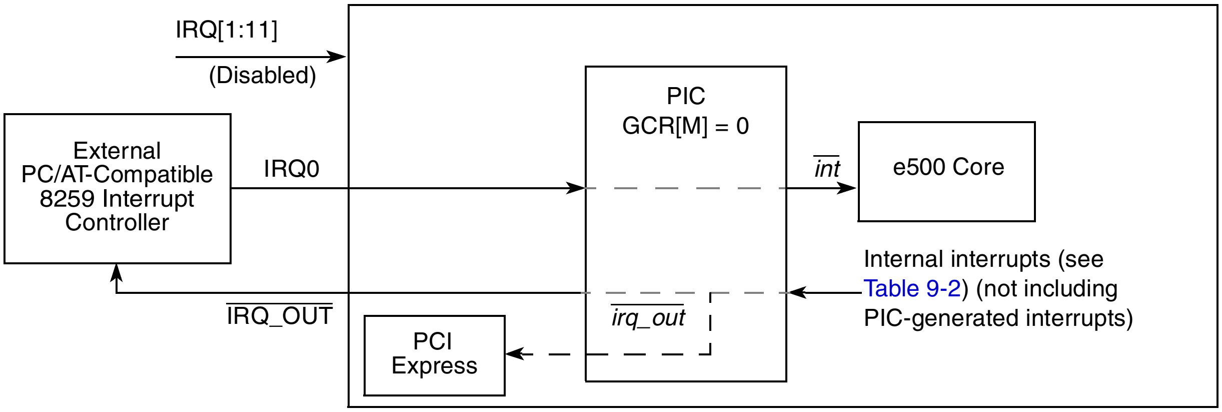

IRQ[1:11]
(Disabled)
External PC/AT-Compatible 8259 Interrupt Controller
IRQ0
PIC
GCR[M] = 0
int
e500 Core
Internal interrupts (see Table 9-2) (not including PIC-generated interrupts)
IRQ_OUT
irq_out
PCI Express

**Figure 9-2. Pass-Through Mode Example**

When pass-through mode is enabled, the internally-generated interrupts shown in Table 9-3 are not forwarded to core 0. Instead, the PIC passes the raw interrupts from the internal sources to $\overline{\text{IRQ\_OUT}}$. Note that when the PCI Express controller is configured as an endpoint (EP) device, the *irq_out* signal from the PIC may used to automatically generate an outbound PCI Express MSI transaction toward the remote interrupt controller resource on the root complex (RC). See Section 17.4.2.1.2, “Hardware MSI Generation.”

Note that in pass-through mode, interrupts generated within the PIC (global timers, interprocessor, and message register interrupts) are disabled. If internal or PIC-generated interrupts must be reported internally to the processor, mixed mode must be used.

## 9.1.4 Interrupt Sources

The PIC can receive separate interrupts from the following sources:

- External—Off-chip signals, IRQ[]
- Internal—On-chip sources from peripheral logic within the integrated device. See Table 9-2.
- Global timers A and B internal to the PIC

- Interprocessor interrupts (IPI)—Intended for communication between different processor cores on the same device. (Can be used for self-interrupt in single-core implementations.)
- Message registers—From within the PIC. Triggered on register write, cleared on read. Used for interprocessor communication.
- Shared message signaled registers—From within the PIC. Triggered on register write, cleared on read. Used for cross-program communication.

### 9.1.4.1 Interrupt Routing—Mixed Mode

When an interrupt request is delivered to the PIC, the corresponding interrupt destination register is checked to determine where the request should be routed, as follows:

- If *x*IDR*n*[EP] = 1 (and all other destination bits are zero), the interrupt is routed off-chip to the external $\overline{\text{IRQ\_OUT}}$ signal. Or if the PCI Express controller is in EP mode and automatically generates a PCI Express MSI transaction. See Section 17.4.2.1.2, “Hardware MSI Generation.”
- If *x*IDR[CI] is set (and all other destination bits are zero), the interrupt is routed to *cint*. .
- If *x*IDR*n*[P0] is set (and all other destination bits are zero) the interrupt is routed to *int0*. Setting *x*IDR*n*[P1] likewise routs the interrupt to *int1* In this case, the interrupt is latched by the interrupt pending register (IPR) and the interrupt flow is as described in Section 9.4.1, “Flow of Interrupt Control.”

### 9.1.4.2 Interrupt Destinations

Following its reset (by default), the PIC directs all timer, shared message signaled, and interrupts from external and internal sources to *int* output (connected to the *int* signal of the processor core).

All other interrupts have more destination options, but only one destination can be chosen for a single interrupt. Instead of being routed to *int*, these interrupts can be routed to the core through $\overline{\text{IRQ\_OUT}}$ or *cint*. These options are selected by writing to the EP or CI fields in the appropriate destination register.

### 9.1.4.3 Internal Interrupt Sources

Table 9-2 shows the assignments of the internal interrupt sources and how they are mapped to the registers that control them. Only the internal interrupts used are listed; that is, the numbers are not consecutive.

**Table 9-2. Internal Interrupt Assignments**

| Internal Interrupt Number | Interrupt Source |
|---|---|
| 0 | L2 cache |
| 1 | ECM |
| 2 | DDR DRAM controller |
| 3 | eLBC controller |
| 4 | DMA 1 channel 1 |
| 5 | DMA 1 channel 2 |
| 6 | DMA 1 channel 3 |
| 7 | DMA 1 channel 4 |

Table 9-2. Internal Interrupt Assignments (continued)

| Internal Interrupt Number | Interrupt Source |
|---|---|
| 8 | PCI |
| 9 | PCI Express Port 2 |
| 10 | PCI Express Port 1 |
| 11 | PCI Express Port 3 |
| 12 | USB 1 controller |
| 13 | eTSEC 1 transmit |
| 14 | eTSEC 1 receive |
| 15 | eTSEC 3 transmit |
| 16 | eTSEC 3 receive |
| 17 | eTSEC 3 error |
| 18 | eTSEC 1 error |
| 19–24 | Reserved |
| 25 | SATA 2 controller |
| 26 | DUART |
| 27 | I$^2$C controllers |
| 28 | Performance monitor |
| 29 | Security interrupt 1 |
| 30 | USB 2 controller |
| 31 | GPIO |
| 32–41 | Reserved |
| 42 | Security interrupt 2 |
| 43 | eSPI |
| 44 | USB 3 controller |
| 45–51 | Reserved |
| 52 | IEEE 1588 eTSEC 1 |
| 53 | Reserved |
| 54 | IEEE 1588 eTSEC 3 |
| 55 | Reserved |
| 56 | eSDHC |
| 57 | Reserved |
| 58 | SATA 1 controller |
| 59–63 | Reserved |

## 9.2 External Signal Descriptions

The following sections describe the PIC signals.

### 9.2.1 Signal Overview

The PIC external interface signals are described in Table 9-3. There are 12 distinct external interrupt request input signals and 1 interrupt request output signal IRQ_OUT. As Table 9-3 shows, the IRQ inputs are also used for delivering INTx signals for the PCI Express root complexes.

### 9.2.2 Detailed Signal Descriptions

Table 9-3 provides detailed descriptions of the external PIC signals.

**Table 9-3. Interrupt Signals—Detailed Signal Descriptions**

| Signal | I/O | Description | |
|---|---|---|---|
| IRQ[0:11] | I | Interrupt request 0–11. The polarity and sense of each of these signals is programmable. All of these inputs can be driven asynchronously.<br>**Note:** Some interrupt request signals IRQ*n* may share PIC external interrupt registers with PCI Express INTx signaling. See Section 9.4.5, "PCI Express INTx/IRQn Sharing." | |
| | | **State Meaning** | Asserted—When an external interrupt signal is asserted (according to the programmed polarity), the PIC checks its priority and the interrupt is conditionally passed to the processor designated in the corresponding destination register. In pass-through mode, only interrupts detected on IRQ0 are passed directly to core 0.<br>Negated—There is no incoming interrupt from that source. |
| | | **Timing** | Assertion—All of these inputs can be asserted asynchronously.<br>Negation—Interrupts programmed as level-sensitive must remain asserted until serviced. Timing requirements for edge-sensitive interrupts can be found in the *Hardware Specifications*. |
| IRQ_OUT | O | Interrupt request out. When the PIC is programmed in pass-through mode, this output reflects the raw interrupts generated by on-chip sources. See Section 9.1.3, "Modes of Operation." | |
| | | **State Meaning** | Asserted—At least one interrupt is currently being signalled to the external system.<br>Negated—Indicates no interrupt source currently routed to IRQ_OUT. |
| | | **Timing** | Because external interrupts are asynchronous with respect to the system clock, both assertion and negation of IRQ_OUT occurs asynchronously with respect to the interrupt source. All timing given here is approximate.<br>Assertion—Internal interrupt source: 2 CCB clock cycles after interrupt occurs.<br>External interrupt source: 4 cycles after interrupt occurs.<br>Message interrupts: 2 cycles after write to message register.<br>Negation—Follows interrupt source negation with the following delay:<br>Internal interrupt: 2 CCB clock cycles<br>External interrupt: 4 cycles.<br>Message interrupts: 2 cycles after message register cleared. |
| MCP | I | Machine check processor. Assertion causes a machine check interrupt to the core. Note that if the core is not configured to process machine check interrupts (MSR[ME] = 0), assertion of MCP causes a core checkstop condition. Note that internal sources for the internal *core_mcp* can also cause a machine check interrupt to the processor core as described in Section 17.4.1.12, "Machine Check Summary Register (MCPSUMR)," Table 9-1 and Table 9-8. | |
| | | **State Meaning** | Asserted—Integrated logic should direct the core to take a machine check interrupt or enter the checkstop state as directed by the MSR.<br>Negated—Machine check handling is not being requested by the external system. |
| | | **Timing** | Assertion—May occur at any time, asynchronous to any clock.<br>Negation—Because MCP*n* is edge-triggered, it can be negated one clock after its assertion. |

## 9.3 Memory Map/Register Definition

The PIC programmable register map occupies 256 Kbytes of memory-mapped space. Reading undefined portions of the memory map returns all zeros; writing has no effect. All PIC registers are 32 bits wide and, although located on 128-bit address boundaries, should be accessed only as 32-bit quantities.

The PIC address offset map, shown in Table 9-4, is divided into three areas:

- 0x*nn*4_0000–0x*nn*4_FFF0—Global registers
- 0x*nn*5_0000–0x*nn*5_FFF0—Interrupt source configuration registers
- 0x*nn*6_0000–0x*nn*6_FFF0—Per-CPU registers

**Table 9-4. PIC Register Address Map**

| Offset | Register | Access | Reset | Section/Page |
|---|---|---|---|---|
| **Global Registers—Block Base Address: 0x4_0000** | | | | |
| 0x0000 | BRR1—Block revision register 1 | R | | 9.3.1.1/9-19 |
| 0x0010 | BRR2—Block revision register 2 | R | 0x0000_0001 | 9.3.1.2/9-19 |
| 0x0014–<br>0x003F | Reserved | — | — | — |
| 0x0040 | IPIDR0—Interprocessor interrupt 0 (IPI 0) dispatch register | W | 0x0000_0000 | 9.3.8.1/9-48 |
| 0x0050 | IPIDR1—IPI 1 dispatch register | | | |
| 0x0060 | IPIDR2—IPI 2 dispatch register | | | |
| 0x0070 | IPIDR3—IPI 3 dispatch register | | | |
| 0x0080 | CTPR—Current task priority register | R/W | 0x0000_000F | 9.3.8.2/9-49 |
| 0x0090 | WHOAMI—Who am I register | R | 0x0000_00*nn* | 9.3.8.3/9-50 |
| 0x00A0 | IACK—Interrupt acknowledge register | R | 0x0000_0000 | 9.3.8.4/9-50 |
| 0x00B0 | EOI—End of interrupt register | W | 0x0000_0000 | 9.3.8.5/9-51 |
| 0x00B4–<br>0x0FFF | Reserved | — | — | — |
| 0x1000 | FRR—Feature reporting register | R | 0x006B_0*n*02 | 9.3.1.3/9-20 |
| 0x1004–0x<br>101F | Reserved | — | — | — |
| 0x1020 | GCR—Global configuration register | R/W | 0x0000_0000 | 9.3.1.4/9-21 |
| 0x1024–x1<br>003F | Reserved | — | — | — |
| 0x1040–<br>0x107F | Vendor reserved | — | — | — |
| 0x1080 | VIR—Vendor identification register | R | 0x0000_0000 | 9.3.1.5/9-21 |
| 0x1090 | PIR—Processor core initialization register | R/W | 0x0000_0000 | 9.3.1.6/9-22 |
| 0x10A0 | IPIVPR0—IPI 0 vector/priority register | R/W | 0x8000_0000 | 9.3.1.7/9-22 |
| 0x10B0 | IPIVPR1—IPI 1 vector/priority register | | | |
| 0x10C0 | IPIVPR2—IPI 2 vector/priority register | | | |
| 0x10D0 | IPIVPR3—IPI 3 vector/priority register | | | |

### Table 9-4. PIC Register Address Map (continued)

| Offset | Register | Access | Reset | Section/Page |
|---|---|---|---|---|
| 0x10E0 | SVR—Spurious vector register | R/W | 0x0000_FFFF | 9.3.1.8/9-23 |
| **Global Timer Group A Registers** | | | | |
| 0x10F0 | TFRRA—Timer frequency reporting register (Group A) | R/W | 0x0000_0000 | 9.3.2.1/9-24 |
| 0x1100 | GTCCRA0—Global timer 0 current count register (Group A) | R | 0x0000_0000 | 9.3.2.2/9-24 |
| 0x1110 | GTBCRA0—Global timer 0 base count register (Group A) | R/W | 0x8000_0000 | 9.3.2.3/9-25 |
| 0x1120 | GTVPRA0—Global timer 0 vector/priority register (Group A) | R/W | 0x8000_0000 | 9.3.2.4/9-25 |
| 0x1130 | GTDRA0—Global timer 0 destination register (Group A) | R/W | 0x0000_0001 | 9.3.2.5/9-26 |
| 0x1140 | GTCCRA1—Global timer 1 current count register (Group A) | R | 0x0000_0000 | 9.3.2.2/9-24 |
| 0x1150 | GTBCRA1—Global timer 1 base count register (Group A) | R/W | 0x8000_0000 | 9.3.2.3/9-25 |
| 0x1160 | GTVPRA1—Global timer 1 vector/priority register (Group A) | R/W | 0x8000_0000 | 9.3.2.4/9-25 |
| 0x1170 | GTDRA1—Global timer 1 destination register (Group A) | R/W | 0x0000_0001 | 9.3.2.5/9-26 |
| 0x1180 | GTCCRA2—Global timer 2 current count register (Group A) | R | 0x0000_0000 | 9.3.2.2/9-24 |
| 0x1190 | GTBCRA2—Global timer 2 base count register (Group A) | R/W | 0x8000_0000 | 9.3.2.3/9-25 |
| 0x11A0 | GTVPRA2—Global timer 2 vector/priority register (Group A) | R/W | 0x8000_0000 | 9.3.2.4/9-25 |
| 0x11B0 | GTDRA2—Global timer 2 destination register (Group A) | R/W | 0x0000_0001 | 9.3.2.5/9-26 |
| 0x11C0 | GTCCRA3—Global timer 3 current count register (Group A) | R | 0x0000_0000 | 9.3.2.2/9-24 |
| 0x11D0 | GTBCRA3—Global timer 3 base count register (Group A) | R/W | 0x8000_0000 | 9.3.2.3/9-25 |
| 0x11E0 | GTVPRA3—Global timer 3 vector/priority register (Group A) | R/W | 0x8000_0000 | 9.3.2.4/9-25 |
| 0x11F0 | GTDRA3—Global timer 3 destination register (Group A) | R/W | 0x0000_0001 | 9.3.2.5/9-26 |
| 0x11F4–<br>0x12FF | Reserved | — | — | — |
| 0x1300 | TCRA—Timer control register (Group A) | R/W | 0x0000_0000 | 9.3.2.6/9-27 |
| 0x1308 | ERQSR—External interrupt summary register | R | 0x0000_0000 | 9.3.3.1/9-29 |
| 0x1310 | IRQSR0—IRQ_OUT summary register 0 | R | 0x0000_0000 | 9.3.3.2/9-29 |
| 0x1320 | IRQSR1—IRQ_OUT summary register 1 | R | 0x0000_0000 | 9.3.3.3/9-30 |
| 0x1324 | IRQSR2—IRQ_OUT summary register 2 | R | 0x0000_0000 | 9.3.3.4/9-31 |
| 0x1330 | CISR0—Critical interrupt summary register 0 | R | 0x0000_0000 | 9.3.3.5/9-31 |
| 0x1340 | CISR1—Critical interrupt summary register 1 | R | 0x0000_0000 | 9.3.3.6/9-32 |
| 0x1344 | CISR2—Critical interrupt summary register 2 | R | 0x0000_0000 | 9.3.3.7/9-32 |
| 0x1350 | PM0MR0—Performance monitor 0 mask register 0 | R/W | 0xFFFF_FFFF | 9.3.4.1/9-33 |
| 0x1360 | PM0MR1—Performance monitor 0 mask register 1 | R/W | 0xFFFF_FFFF | 9.3.4.2/9-34 |
| 0x1364 | PM0MR2—Performance monitor 0 mask register 2 | R/W | 0xFFFF_FFFF | 9.3.4.2/9-34 |
| 0x1370 | PM1MR0—Performance monitor 1 mask register 0 | R/W | 0xFFFF_FFFF | 9.3.4.1/9-33 |
| 0x1380 | PM1MR1—Performance monitor 1 mask register 1 | R/W | 0xFFFF_FFFF | 9.3.4.2/9-34 |
| 0x1384 | PM1MR2—Performance monitor 1 mask register 2 | R/W | 0xFFFF_FFFF | 9.3.4.2/9-34 |
| 0x1390 | PM2MR0—Performance monitor 2 mask register 0 | R/W | 0xFFFF_FFFF | 9.3.4.1/9-33 |
| 0x13A0 | PM2MR1—Performance monitor 2 mask register 1 | R/W | 0xFFFF_FFFF | 9.3.4.2/9-34 |

**Table 9-4. PIC Register Address Map (continued)**

| Offset | Register | Access | Reset | Section/Page |
|---|---|---|---|---|
| 0x13A4 | PM2MR2—Performance monitor 2 mask register 2 | R/W | 0xFFFF_FFFF | 9.3.4.2/9-34 |
| 0x13B0 | PM3MR0—Performance monitor 3 mask register 0 | R/W | 0xFFFF_FFFF | 9.3.4.1/9-33 |
| 0x13C0 | PM3MR1—Performance monitor 3 mask register 1 | R/W | 0xFFFF_FFFF | 9.3.4.2/9-34 |
| 0x13C4 | PM3MR2—Performance monitor 3 mask register 2 | R/W | 0xFFFF_FFFF | 9.3.4.2/9-34 |
| 0x13C8–0x13FF | Reserved | — | — | — |
| 0x1400 | MSGR0—Message register 0 | R/W | 0x0000_0000 | 9.3.5.1/9-35 |
| 0x1410 | MSGR1—Message register 1 | R/W | 0x0000_0000 | 9.3.5.1/9-35 |
| 0x1420 | MSGR2—Message register 2 | R/W | 0x0000_0000 | 9.3.5.1/9-35 |
| 0x1430 | MSGR3—Message register 3 | R/W | 0x0000_0000 | 9.3.5.1/9-35 |
| 0x1434–0x14FF | Reserved | — | — | — |
| 0x1500 | MER—Message enable register | R/W | 0x0000_0000 | 9.3.5.2/9-35 |
| 0x1510 | MSR—Message status register | R/W | 0x0000_0000 | 9.3.5.3/9-36 |
| 0x1514–0x15FF | Reserved | — | — | — |
| 0x1600 | MSIR0—Shared message signaled interrupt register 0 | RC | 0x0000_0000 | 9.3.6.1/9-37 |
| 0x1610 | MSIR1—Shared message signaled interrupt register 1 | RC | 0x0000_0000 | 9.3.6.1/9-37 |
| 0x1620 | MSIR2—Shared message signaled interrupt register 2 | RC | 0x0000_0000 | 9.3.6.1/9-37 |
| 0x1630 | MSIR3—Shared message signaled interrupt register 3 | RC | 0x0000_0000 | 9.3.6.1/9-37 |
| 0x1640 | MSIR4—Shared message signaled interrupt register 4 | RC | 0x0000_0000 | 9.3.6.1/9-37 |
| 0x1650 | MSIR5—Shared message signaled interrupt register 5 | RC | 0x0000_0000 | 9.3.6.1/9-37 |
| 0x1660 | MSIR6—Shared message signaled interrupt register 6 | RC | 0x0000_0000 | 9.3.6.1/9-37 |
| 0x1670 | MSIR7—Shared message signaled interrupt register 7 | RC | 0x0000_0000 | 9.3.6.1/9-37 |
| 0x1674–0x171F | Reserved | — | — | — |
| 0x1720 | MSISR—Shared message signaled interrupt status register | R | 0x0000_0000 | 9.3.6.2/9-37 |
| 0x1740 | MSIIR—Shared message signaled interrupt index register | W | 0x0000_0000 | 9.3.6.3/9-38 |
| 0x1744–0x20EF | Reserved | — | — | — |
| **Global Timer Group B Registers** | | | | |
| 0x20F0 | TFRRB—Timer frequency reporting register group B | R/W | 0x0000_0000 | 9.3.2.1/9-24 |
| 0x2100 | GTCCRB0—Global timer current count register group B 0 | R | 0x0000_0000 | 9.3.2.2/9-24 |
| 0x2110 | GTBCRB0—Global timer base count register group B 0 | R/W | 0x8000_0000 | 9.3.2.3/9-25 |
| 0x2120 | GTVPRB0—Global timer vector/priority register group B 0 | R/W | 0x8000_0000 | 9.3.2.4/9-25 |
| 0x2130 | GTDRB0—Global timer destination register group B 0 | R/W | 0x0000_0001 | 9.3.2.5/9-26 |
| 0x2140 | GTCCRB1—Global timer current count register group B 1 | R | 0x0000_0000 | 9.3.2.2/9-24 |
| 0x2150 | GTBCRB1—Global timer base count register group B 1 | R/W | 0x8000_0000 | 9.3.2.3/9-25 |

Table 9-4. PIC Register Address Map (continued)

| Offset | Register | Access | Reset | Section/Page |
|---|---|---|---|---|
| 0x2160 | GTVPRB1—Global timer vector/priority register group B 1 | R/W | 0x8000_0000 | 9.3.2.4/9-25 |
| 0x2170 | GTDRB1—Global timer destination register group B 1 | R/W | 0x0000_0001 | 9.3.2.5/9-26 |
| 0x2180 | GTCCRB2—Global timer current count register group B 2 | R | 0x0000_0000 | 9.3.2.2/9-24 |
| 0x2190 | GTBCRB2—Global timer base count register group B 2 | R/W | 0x8000_0000 | 9.3.2.3/9-25 |
| 0x21A0 | GTVPRB2—Global timer vector/priority register group B 2 | R/W | 0x8000_0000 | 9.3.2.4/9-25 |
| 0x21B0 | GTDRB2—Global timer destination register group B 2 | R/W | 0x0000_0001 | 9.3.2.5/9-26 |
| 0x21C0 | GTCCRB3—Global timer current count register group B 3 | R | 0x0000_0000 | 9.3.2.2/9-24 |
| 0x21D0 | GTBCRB3—Global timer base count register group B 3 | R/W | 0x8000_0000 | 9.3.2.3/9-25 |
| 0x21E0 | GTVPRB3—Global timer vector/priority register group B 3 | R/W | 0x8000_0000 | 9.3.2.4/9-25 |
| 0x21F0 | GTDRB3—Global timer destination register group B 3 | R/W | 0x0000_0001 | 9.3.2.5/9-26 |
| 0x21F4–<br>0x22FF | Reserved | — | — | — |
| 0x2300 | TCRB—Timer control register (Group B) | R/W | 0x0000_0000 | 9.3.2.6/9-27 |
| 0x2304–<br>0x23FF | Reserved | — | — | — |
| 0x2400 | MSGR4—Message register 4 | R/W | 0x0000_0000 | 9.3.5.1/9-35 |
| 0x2410 | MSGR5—Message register 5 | | | |
| 0x2420 | MSGR6—Message register 6 | | | |
| 0x2430 | MSGR7—Message register 7 | | | |
| 0x2434–<br>0x24FF | Reserved | — | — | — |
| 0x2500 | MER—Message enable register (for MSGR4–7) | R/W | 0x0000_0000 | 9.3.5.2/9-35 |
| 0x2510 | MSR—Message status register (for MSGG4–7) | R/W | 0x0000_0000 | 9.3.5.3/9-36 |
| 0x2514–<br>0xFFFF | Reserved | — | — | — |
| **Interrupt Source Configuration Registers—Block Base Address: 0x5_0000** | | | | |
| 0x0000 | EIVPR0—External interrupt 0 (IRQ0) vector/priority register or PEX1-INTA vector/priority register | R/W | 0x8000_0000 | 9.3.7.1/9-41 |
| 0x0010 | EIDR0—External interrupt 0 (IRQ0) destination register or PEX1-INTA destination register | R/W | 0x0000_0001 | 9.3.7.2/9-42 |
| 0x0020 | EIVPR1—External interrupt 1 (IRQ1) vector/priority register or PEX1-INTB vector/priority register | R/W | 0x8000_0000 | 9.3.7.1/9-41 |
| 0x0030 | EIDR1—External interrupt 1 (IRQ1) destination register or PEX1-INTB destination register | R/W | 0x0000_0001 | 9.3.7.2/9-42 |
| 0x0040 | EIVPR2—External interrupt 2 (IRQ2) vector/priority register or PEX1-INTC vector/priority register | R/W | 0x8000_0000 | 9.3.7.1/9-41 |
| 0x0050 | EIDR2—External interrupt 2 (IRQ2) destination register or PEX1-INTC destination register | R/W | 0x0000_0001 | 9.3.7.2/9-42 |
| 0x0060 | EIVPR3—External interrupt 3 (IRQ3) vector/priority register or PEX1-INTD vector/priority register | R/W | 0x8000_0000 | 9.3.7.1/9-41 |

Table 9-4. PIC Register Address Map (continued)

| Offset | Register | Access | Reset | Section/Page |
|---|---|---|---|---|
| 0x0070 | EIDR3—External interrupt 3 (IRQ3) destination register or PEX1-INTD destination register | R/W | 0x0000_0001 | 9.3.7.2/9-42 |
| 0x0080 | EIVPR4—External interrupt 4 (IRQ4) vector/priority register or PEX2-INTA vector/priority register | R/W | 0x8000_0000 | 9.3.7.1/9-41 |
| 0x0090 | EIDR4—External interrupt 4 (IRQ4) destination register or PEX2-INTA destination register | R/W | 0x0000_0001 | 9.3.7.2/9-42 |
| 0x00A0 | EIVPR5—External interrupt 5 (IRQ5) vector/priority register or PEX2-INTB vector/priority register | R/W | 0x8000_0000 | 9.3.7.1/9-41 |
| 0x00B0 | EIDR5—External interrupt 5 (IRQ5) destination register or PEX2-INTB destination register | R/W | 0x0000_0001 | 9.3.7.2/9-42 |
| 0x00C0 | EIVPR6—External interrupt 6 (IRQ6) vector/priority register or PEX2-INTC vector/priority register | R/W | 0x8000_0000 | 9.3.7.1/9-41 |
| 0x00D0 | EIDR6—External interrupt 6 (IRQ6) destination register or PEX2-INTC destination register | R/W | 0x0000_0001 | 9.3.7.2/9-42 |
| 0x00E0 | EIVPR7—External interrupt 7 (IRQ7) vector/priority register or PEX2-INTD vector/priority register | R/W | 0x8000_0000 | 9.3.7.1/9-41 |
| 0x00F0 | EIDR7—External interrupt 7 (IRQ7) destination register or PEX2-INTD destination register | R/W | 0x0000_0001 | 9.3.7.2/9-42 |
| 0x0100 | EIVPR8—External interrupt 8 (IRQ8) vector/priority register or PEX3-INTA vector/priority register | R/W | 0x8000_0000 | 9.3.7.1/9-41 |
| 0x0110 | EIDR8—External interrupt 8 (IRQ8) destination register or PEX3-INTA destination register | R/W | 0x0000_0001 | 9.3.7.2/9-42 |
| 0x0120 | EIVPR9—External interrupt 9 (IRQ9) vector/priority register or PEX3-INTB vector/priority register | R/W | 0x8000_0000 | 9.3.7.1/9-41 |
| 0x0130 | EIDR9—External interrupt 9 (IRQ9) destination register or PEX3-INTB destination register | R/W | 0x0000_0001 | 9.3.7.2/9-42 |
| 0x0140 | EIVPR10—External interrupt 10 (IRQ10) vector/priority register or PEX3-INTC vector/priority register | R/W | 0x8000_0000 | 9.3.7.1/9-41 |
| 0x0150 | EIDR10—External interrupt 10 (IRQ10) destination register or PEX3-INTC destination register | R/W | 0x0000_0001 | 9.3.7.2/9-42 |
| 0x0160 | EIVPR11—External interrupt 11 (IRQ11) vector/priority register or PEX3-INTD vector/priority register | R/W | 0x8000_0000 | 9.3.7.1/9-41 |
| 0x0170 | EIDR11—External interrupt 11 (IRQ11) destination register or PEX3-INTD destination register | R/W | 0x0000_0001 | 9.3.7.2/9-42 |
| 0x0174–0x01FF | Reserved | — | — | — |
| 0x0200 | IIVPR0—Internal interrupt 0 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0210 | IIDR0—Internal interrupt 0 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0220 | IIVPR1—Internal interrupt 1 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0230 | IIDR1—Internal interrupt 1 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0240 | IIVPR2—Internal interrupt 2 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0250 | IIDR2—Internal interrupt 2 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0260 | IIVPR3—Internal interrupt 3 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |

Table 9-4. PIC Register Address Map (continued)

| Offset | Register | Access | Reset | Section/Page |
|---|---|---|---|---|
| 0x0270 | IIDR3—Internal interrupt 3 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0280 | IIVPR4—Internal interrupt 4 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0290 | IIDR4—Internal interrupt 4 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x02A0 | IIVPR5—Internal interrupt 5 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x02B0 | IIDR5—Internal interrupt 5 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x02C0 | IIVPR6—Internal interrupt 6 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x02D0 | IIDR6—Internal interrupt 6 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x02E0 | IIVPR7—Internal interrupt 7 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x02F0 | IIDR7—Internal interrupt 7 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0300 | IIVPR8—Internal interrupt 8 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0310 | IIDR8—Internal interrupt 8 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0320 | IIVPR9—Internal interrupt 9 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0330 | IIDR9—Internal interrupt 9 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0340 | IIVPR10—Internal interrupt 10 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0350 | IIDR10—Internal interrupt 10 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0360 | IIVPR11—Internal interrupt 11 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0370 | IIDR11—Internal interrupt 11 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0380 | IIVPR12—Internal interrupt 12 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0390 | IIDR12—Internal interrupt 12 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x03A0 | IIVPR13—Internal interrupt 13 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x03B0 | IIDR13—Internal interrupt 13 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x03C0 | IIVPR14—Internal interrupt 14 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x03D0 | IIDR14—Internal interrupt 14 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x03E0 | IIVPR15—Internal interrupt 15 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x03F0 | IIDR15—Internal interrupt 15 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0400 | IIVPR16—Internal interrupt 16 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0410 | IIDR16—Internal interrupt 16 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0420 | IIVPR17—Internal interrupt 17 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0430 | IIDR17—Internal interrupt 17 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0440 | IIVPR18—Internal interrupt 18 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0450 | IIDR18—Internal interrupt 18 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0460 | IIVPR19—Internal interrupt 19 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0470 | IIDR19—Internal interrupt 19 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0480 | IIVPR20—Internal interrupt 20 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0490 | IIDR20—Internal interrupt 20 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x04A0 | IIVPR21—Internal interrupt 21 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x04B0 | IIDR21—Internal interrupt 21 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |

Table 9-4. PIC Register Address Map (continued)

| Offset | Register | Access | Reset | Section/Page |
|---|---|---|---|---|
| 0x04C0 | IIVPR22—Internal interrupt 22 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x04D0 | IIDR22—Internal interrupt 22 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x04E0 | IIVPR23—Internal interrupt 23 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x04F0 | IIDR23—Internal interrupt 23 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0500 | IIVPR24—Internal interrupt 24 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0510 | IIDR24—Internal interrupt 24 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0520 | IIVPR25—Internal interrupt 25 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0530 | IIDR25—Internal interrupt 25 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0540 | IIVPR26—Internal interrupt 26 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0550 | IIDR26—Internal interrupt 26 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0560 | IIVPR27—Internal interrupt 27 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0570 | IIDR27—Internal interrupt 27 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0580 | IIVPR28—Internal interrupt 28 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0590 | IIDR28—Internal interrupt 28 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x05A0 | IIVPR29—Internal interrupt 29 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x05B0 | IIDR29—Internal interrupt 29 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x05C0 | IIVPR30—Internal interrupt 30 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x05D0 | IIDR30—Internal interrupt 30 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x05E0 | IIVPR31—Internal interrupt 31 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x05F0 | IIDR31—Internal interrupt 31 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0600 | IIVPR32—Internal interrupt 32 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0610 | IIDR32—Internal interrupt 32 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0620 | IIVPR33—Internal interrupt 33 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0630 | IIDR33—Internal interrupt 33 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0640 | IIVPR34—Internal interrupt 34 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0650 | IIDR34—Internal interrupt 34 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0660 | IIVPR35—Internal interrupt 35 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0670 | IIDR35—Internal interrupt 35 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0680 | IIVPR36—Internal interrupt 36 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0690 | IIDR36—Internal interrupt 36 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x06A0 | IIVPR37—Internal interrupt 37 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x06B0 | IIDR37—Internal interrupt 37 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x06C0 | IIVPR38—Internal interrupt 38 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x06D0 | IIDR38—Internal interrupt 38 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x06E0 | IIVPR39—Internal interrupt 39 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x06F0 | IIDR39—Internal interrupt 39 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0700 | IIVPR40—Internal interrupt 40 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |

Table 9-4. PIC Register Address Map (continued)

| Offset | Register | Access | Reset | Section/Page |
|---|---|---|---|---|
| 0x0710 | IIDR40—Internal interrupt 40 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0720 | IIVPR41—Internal interrupt 41 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0730 | IIDR41—Internal interrupt 41 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0740 | IIVPR42—Internal interrupt 42 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0750 | IIDR42—Internal interrupt 42 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0760 | IIVPR43—Internal interrupt 43 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0770 | IIDR43—Internal interrupt 43 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0780 | IIVPR44—Internal interrupt 44 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0790 | IIDR44—Internal interrupt 44 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x07A0 | IIVPR45—Internal interrupt 45 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x07B0 | IIDR45—Internal interrupt 45 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x07C0 | IIVPR46—Internal interrupt 46 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x07D0 | IIDR46—Internal interrupt 46 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x07E0 | IIVPR47—Internal interrupt 47 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x07F0 | IIDR47—Internal interrupt 47 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0800 | IIVPR48—Internal interrupt 48 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0810 | IIDR48—Internal interrupt 48 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0820 | IIVPR49—Internal interrupt 49 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0830 | IIDR49—Internal interrupt 49 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0840 | IIVPR50—Internal interrupt 50 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0850 | IIDR50—Internal interrupt 50 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0860 | IIVPR51—Internal interrupt 51 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0870 | IIDR51—Internal interrupt 51 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0880 | IIVPR52—Internal interrupt 52 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0890 | IIDR52—Internal interrupt 52 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x08A0 | IIVPR53—Internal interrupt 53 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x08B0 | IIDR53—Internal interrupt 53 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x08C0 | IIVPR54—Internal interrupt 54 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x08D0 | IIDR54—Internal interrupt 54 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x08E0 | IIVPR55—Internal interrupt 55 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x08F0 | IIDR55—Internal interrupt 55 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0900 | IIVPR56—Internal interrupt 56 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0910 | IIDR56—Internal interrupt 56 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0920 | IIVPR57—Internal interrupt 57 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0930 | IIDR57—Internal interrupt 57 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0940 | IIVPR58—Internal interrupt 58 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0950 | IIDR58—Internal interrupt 58 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |

### Table 9-4. PIC Register Address Map (continued)

| Offset | Register | Access | Reset | Section/Page |
|---|---|---|---|---|
| 0x0960 | IIVPR59—Internal interrupt 59 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0970 | IIDR59—Internal interrupt 59 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0980 | IIVPR60—Internal interrupt 60 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0990 | IIDR60—Internal interrupt 60 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x09A0 | IIVPR61—Internal interrupt 61 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x09B0 | IIDR61—Internal interrupt 61 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x09C0 | IIVPR62—Internal interrupt 62 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x09D0 | IIDR62—Internal interrupt 62 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x09E0 | IIVPR63—Internal interrupt 63 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x09F0 | IIDR63—Internal interrupt 63 destination register | R/W | 0x0000_0001 | 9.3.7.3/9-43 |
| 0x09F4–0x15FF | Reserved | — | — | — |
| 0x1600 | MIVPR0—Messaging interrupt 0 (MSG 0) vector/priority register | R/W | 0x8000_0000 | 9.3.7.5/9-45 |
| 0x1610 | MIDR0—Messaging interrupt 0 (MSG 0) destination register | R/W | 0x0000_0001 | 9.3.7.6/9-46 |
| 0x1620 | MIVPR1—Messaging interrupt 1 (MSG 1) vector/priority register | R/W | 0x8000_0000 | 9.3.7.5/9-45 |
| 0x1630 | MIDR1—Messaging interrupt 1 (MSG 1) destination register | R/W | 0x0000_0001 | 9.3.7.6/9-46 |
| 0x1640 | MIVPR2—Messaging interrupt 2 (MSG 2) vector/priority register | R/W | 0x8000_0000 | 9.3.7.5/9-45 |
| 0x1650 | MIDR2—Messaging interrupt 2 (MSG 2) destination register | R/W | 0x0000_0001 | 9.3.7.6/9-46 |
| 0x1660 | MIVPR3—Messaging interrupt 3 (MSG 3) vector/priority register | R/W | 0x8000_0000 | 9.3.7.5/9-45 |
| 0x1670 | MIDR3—Messaging interrupt 3 (MSG 3) destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x1674 | MIVPR4—Messaging interrupt 4 (MSG 4) vector/priority register | R/W | 0x8000_0000 | 9.3.7.5/9-45 |
| 0x1690 | MIDR4—Messaging interrupt 4 (MSG 4) destination register | R/W | 0x0000_0001 | 9.3.7.6/9-46 |
| 0x16A0 | MIVPR5—Messaging interrupt 5 (MSG 5) vector/priority register | R/W | 0x8000_0000 | 9.3.7.5/9-45 |
| 0x16B0 | MIDR5—Messaging interrupt 5 (MSG 5) destination register | R/W | 0x0000_0001 | 9.3.7.6/9-46 |
| 0x16C0 | MIVPR6—Messaging interrupt 6 (MSG 6) vector/priority register | R/W | 0x8000_0000 | 9.3.7.5/9-45 |
| 0x16D0 | MIDR6—Messaging interrupt 6 (MSG 6) destination register | R/W | 0x0000_0001 | 9.3.7.6/9-46 |
| 0x16E0 | MIVPR7—Messaging interrupt 7 (MSG 7) vector/priority register | R/W | 0x8000_0000 | 9.3.7.5/9-45 |
| 0x16F0 | MIDR7—Messaging interrupt 7 (MSG 7) destination register | R/W | 0x0000_0001 | 9.3.7.6/9-46 |
| 0x16F4–0x1BFF | Reserved | — | — | — |
| 0x1C00 | MSIVPR0—Shared message signaled interrupt vector/priority register 0 | R/W | 0x8000_0000 | 9.3.6.4/9-38 |
| 0x1C10 | MSIDR0—Shared message signaled interrupt destination register 0 | R/W | 0x0000_0001 | 9.3.6.5/9-39 |
| 0x1C20 | MSIVPR1—Shared message signaled interrupt vector/priority register 1 | R/W | 0x8000_0000 | 9.3.6.4/9-38 |
| 0x1C30 | MSIDR1—Shared message signaled interrupt destination register 1 | R/W | 0x0000_0001 | 9.3.6.5/9-39 |
| 0x1C40 | MSIVPR2—Shared message signaled interrupt vector/priority register 2 | R/W | 0x8000_0000 | 9.3.6.4/9-38 |
| 0x1C50 | MSIDR2—Shared message signaled interrupt destination register 2 | R/W | 0x0000_0001 | 9.3.6.5/9-39 |
| 0x1C60 | MSIVPR3—Shared message signaled interrupt vector/priority register 3 | R/W | 0x8000_0000 | 9.3.6.4/9-38 |
| 0x1C70 | MSDIR3—Shared message signaled interrupt destination register 3 | R/W | 0x0000_0001 | 9.3.6.5/9-39 |

Table 9-4. PIC Register Address Map (continued)

| Offset | Register | Access | Reset | Section/Page |
|---|---|---|---|---|
| 0x1C80 | MSIVPR4—Shared message signaled interrupt vector/priority register 4 | R/W | 0x8000_0000 | 9.3.6.4/9-38 |
| 0x1C90 | MSIDR4—Shared message signaled interrupt destination register 4 | R/W | 0x0000_0001 | 9.3.6.5/9-39 |
| 0x1CA0 | MSIVPR5—Shared message signaled interrupt vector/priority register 5 | R/W | 0x8000_0000 | 9.3.6.4/9-38 |
| 0x1CB0 | MSIDR5—Shared message signaled interrupt destination register 5 | R/W | 0x0000_0001 | 9.3.6.5/9-39 |
| 0x1CC0 | MSIVPR6—Shared message signaled interrupt vector/priority register 6 | R/W | 0x8000_0000 | 9.3.6.4/9-38 |
| 0x1CD0 | MSIDR6—Shared message signaled interrupt destination register 6 | R/W | 0x0000_0001 | 9.3.6.5/9-39 |
| 0x1CE0 | MSIVPR7—Shared message signaled interrupt vector/priority register 7 | R/W | 0x8000_0000 | 9.3.6.4/9-38 |
| 0x1CF0 | MSDIR7—Shared message signaled interrupt destination register 7 | R/W | 0x0000_0001 | 9.3.6.5/9-39 |
| 0x1CFF–<br>0xFFFF | Reserved | — | — | — |
| **Per-CPU Registers Block Base Address: 0x6_0000** | | | | |
| 0x0000–<br>0x003F | Reserved | — | — | — |
| 0x0040 | IPIDR0—Processor core 0 interprocessor 0 dispatch register | W | 0x0000_0000 | 9.3.8.1/9-48 |
| 0x0050 | IPIDR1—Processor core 0 interprocessor 1 dispatch register | | | |
| 0x0060 | IPIDR2—Processor core 0 interprocessor 2 dispatch register | | | |
| 0x0070 | IPIDR3—Processor core 0 interprocessor 3 dispatch register | | | |
| 0x0080 | CTPR0—Processor core 0 current task priority register | R/W | 0x0000_000F | 9.3.8.2/9-49 |
| 0x0090 | WHOAMI0—Processor core 0 who am I register | R | 0x0000_00*nn* | 9.3.8.3/9-50 |
| 0x00A0 | IACK0—Processor core 0 interrupt acknowledge register | R | 0x0000_0000 | 9.3.8.4/9-50 |
| 0x00B0 | EOI0—Processor core 0 end of interrupt register | W | 0x0000_0000 | 9.3.8.5/9-51 |
| 0x00BF–<br>0x0FFF | Reserved | — | — | — |
| 0x1000–<br>0x103F | Reserved | — | — | — |
| 0x1040 | IPIDR0—Processor core 1 interprocessor 0 dispatch register | W | 0x0000_0000 | 9.3.8.1/9-48 |
| 0x1050 | IPIDR1—Processor core 1 interprocessor 1 dispatch register | | | |
| 0x1060 | IPIDR2—Processor core 1 interprocessor 2 dispatch register | | | |
| 0x1070 | IPIDR3—Processor core 1 interprocessor 3 dispatch register | | | |
| 0x1080 | CTPR1—Processor core 1 current task priority register | R/W | 0x0000_000F | 9.3.8.2/9-49 |
| 0x1090 | WHOAMI1—Processor core 1 who am I register | R | n/a | 9.3.8.3/9-50 |
| 0x10A0 | IACK1—Processor core 1 interrupt acknowledge register | R | 0x0000_0000 | 9.3.8.4/9-50 |
| 0x10B0 | EOI1—Processor core 1 end of interrupt register | W | 0x0000_0000 | 9.3.8.5/9-51 |
| 0x10BF–<br>0xFFFF | Reserved | — | — | — |

### 9.3.1 Global Registers

Although most PIC registers have one address, some are replicated for each processor core in a multiprocessor device. For such registers, each core accesses its separate registers using the same address, the address decoding being sensitive to the processor core ID. A copy of the per-CPU registers is available to each processor core at the same physical address, that is, in a private access address space that acts like an alias to a processor's own copy of the per-CPU registers. As shown in Figure 9-44, the ID of the core initiating the read/write transaction determines which processor's per-CPU registers to access. For more information, see Section 9.3.8, "Per-CPU (Private Access) Registers."

**NOTE**

> Register fields designated as write-1-to-clear are cleared only by writing ones to them. Writing zeros to them has no effect.

#### 9.3.1.1 Block Revision Register 1 (BRR1)

BRR1, shown in Figure 9-3, provides information about the PIC IP block.

Offset 0x0000 Access: Read only

| | 0 … 15 | 16 … 23 | 24 … 31 |
|---|---|---|---|
| R | IPID | IPMJ | IPMN |
| W | | | |

**Figure 9-3. Block Revision Register 1 (BRR1)**

Table 9-7 describes the BRR1 fields.

**Table 9-5. BRR1 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–15 | IPID | IP block ID. |
| 16–23 | IPMJ | The major revision of the IP block. |
| 24–31 | IPMN | The minor revision of the IP block. |

#### 9.3.1.2 Block Revision Register 2 (BRR2)

BRR2, shown in Figure 9-4, provides information about the IP block integration option and IP block configuration options.

Offset 0x0010 Access: Read only

| | 0 … 7 | 8 … 15 | 16 … 23 | 24 … 31 |
|---|---|---|---|---|
| R | — | IPINTO | — | IIPCFGO |
| W | | | | |
| Reset | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 1 |

**Figure 9-4. Block Revision Register 2 (BRR2)**

Table 9-6 describes the BRR2 fields.

Table 9-6. BRR2 Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–7 | — | Reserved, should be cleared. |
| 8–15 | IPINTO | IP block integration options |
| 16–23 | — | Reserved, should be cleared. |
| 24–31 | IPCFGO | IP block configuration options |

### 9.3.1.3 Feature Reporting Register (FRR)

FRR, shown in Figure 9-5, provides information about interrupt and processor core configurations. It also informs the programming environment of the controller version.

Offset 0x1000 Access: Read only

| | 0–3 | 4–15 | 16–18 | 19–23 | 24–31 |
|---|---|---|---|---|---|
| R | — | NIRQ | — | NCPU | VID |
| W | | | | | |
| Reset | 0 0 0 0 | 0 0 0 0 0 1 1 0 0 1 1 1 | 0 0 0 | 0 0 0 0 *n* | 0 0 0 0 0 0 1 0 |

Figure 9-5. Feature Reporting Register (FRR)

Table 9-7 describes the FRR fields.

Table 9-7. FRR Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–4 | — | Reserved, should be cleared. |
| 5–15 | NIRQ | Number of interrupts. Holds the binary value of the number of the highest interrupt source supported minus one. The value is 103 (0x67 or 0b000_0110_0111), because this device supports 104 interrupts: 12 external sources, 64 internal sources (see Table 9-3), 8 timer sources, 8 interprocessor sources, 4 messaging sources, and 8 shared message signaled sources.A zero in this field corresponds to one source. |
| 16–18 | — | Reserved, should be cleared |
| 19–23 | NCPU | Number of CPUs. The number of the highest physical CPUs (or processor cores) supported minus one.<br>00000 Single core—core0<br>00001 Two cores—core0 and core1 |
| 24–31 | VID | Version ID. Reports the OpenPIC specification revision level supported by this interrupt controller implementation. The VID field's value of two (0x02) corresponds to revision 1.2 which is the revision level currently supported. |

### 9.3.1.4 Global Configuration Register (GCR)

GCR, shown in Figure 9-6, controls the PIC's operating mode, and allows software to reset the PIC.

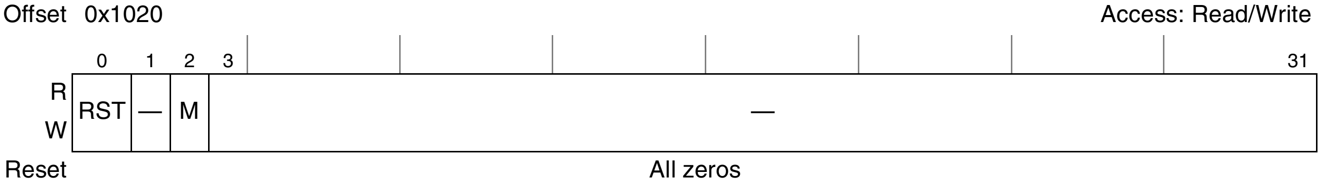


**Figure 9-6. Global Configuration Register (GCR)**

Table 9-8 describes the GCR fields.

**Table 9-8. GCR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | RST | Reset. Setting RST forces the PIC to be reset. Cleared automatically when the reset sequence is complete. See Section 9.4.8, "Resetting the PIC," for more information. |
| 1 | — | Reserved, should be cleared. |
| 2 | M | Mode. PIC operating mode. Section 9.1.3, "Modes of Operation," provides details about these modes.<br>0 Pass-through mode. On-chip PIC is disabled and interrupts detected on IRQ0 are passed directly to core 0.<br>1 Mixed mode. Interrupts are handled by the normal priority and delivery mechanisms of the PIC. |
| 3–31 | — | Reserved, should be cleared. |

### 9.3.1.5 Vendor Identification Register (VIR)

VIR, shown in Figure 9-7, is defined by the OpenPIC specifications and is provided for compliance. The zero value for VIR[VENDORID] indicates a generic OpenPIC-compliant device, which makes the other VIR fields meaningless.

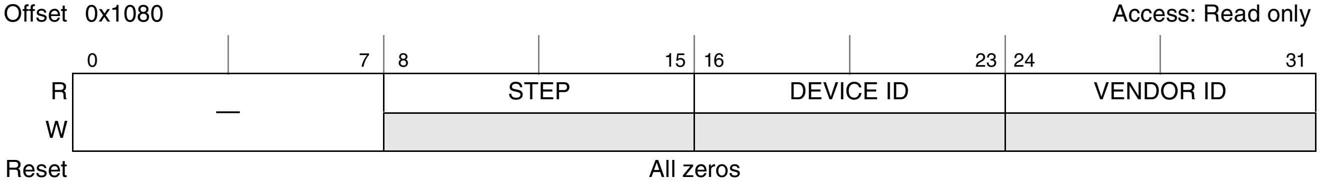


**Figure 9-7. Vendor Identification Register (VIR)**

Table 9-9 describes the VIR fields.

**Table 9-9. VIR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–7 | — | Reserved, should be cleared. |
| 8–15 | STEP | Stepping. Indicates the silicon revision for this device. Has no meaning if VENDOR ID value is zero. |

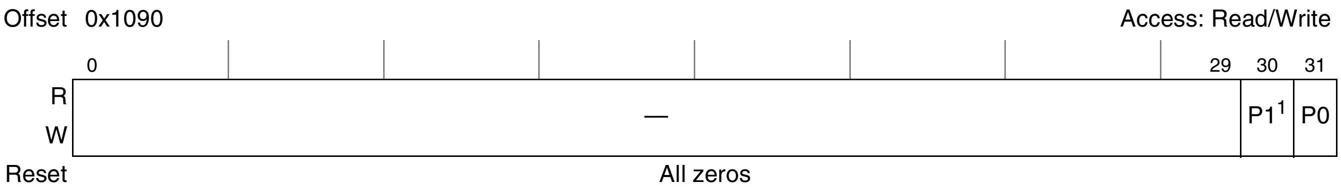
**Table 9-9. VIR Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 16–23 | DEVICE ID | Device identification. Vendor-specified identifier for this device. Has no meaning if VENDOR ID is zero. |
| 24–31 | VENDOR ID | Vendor identification. Specifies the manufacturer of this part. A value of zero implies a generic OpenPIC-compliant device. |

### 9.3.1.6 Processor Core Initialization Register (PIR)

PIR, shown in Figure 9-8, provides a way for software to generate a core reset. Setting P1 or P0 causes the respective *core0_hreset* or *core1_hreset* signal to assert. Note that after requesting a core reset using this register the applicable bit should not be cleared until the requested core reset has occurred.

Note that although the OpenPIC architecture was defined to support up to 32 processing cores, only fields corresponding to the number of cores on the device are implemented.



[1] Reserved in single-processor implementations.

**Figure 9-8. Processor Core Initialization Register (PIR)**

Table 9-10 describes the PIR fields.

**Table 9-10. PIR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–29 | — | Reserved, should be cleared. |
| 30 | P1 | Processor core 1 reset. Setting this bit causes the PIC to assert the *core1_hreset* signal. Reserved in single-processor implementations. |
| 31 | P0 | Processor core 0 reset. Setting this bit causes the PIC to assert the *core0_hreset* signal. |

### 9.3.1.7 Interprocessor Interrupt Vector/Priority Registers (IPIVPR0–IPIVPR3)

IPIVPRs, shown in Figure 9-9, contain the interrupt vector and priority fields for the four interprocessor interrupt channels. There is one vector/priority register per channel. The VECTOR and PRIORITY values should not be changed while IPIVPR*n*[A] is set. See Section 9.4.1, “Flow of Interrupt Control,” for information on IPR and ISR.

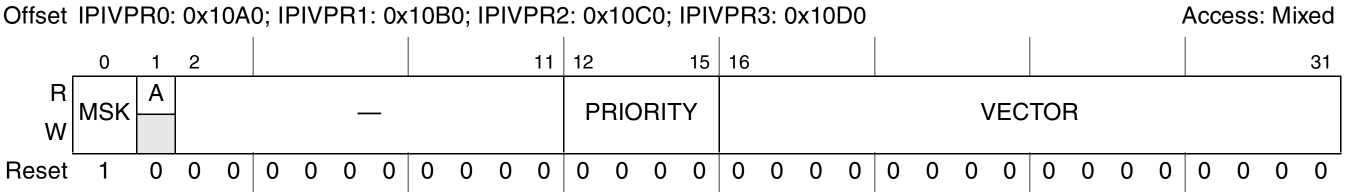


**Figure 9-9. Interprocessor Interrupt Vector/Priority Register (IPIVPR*n*)**

Table 9-11 describes the IPIVPR*n* fields.

**Table 9-11. IPIVPR*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | MSK | Mask. Mask interrupts to *int* from this source.<br>0 An interrupt request is generated if the corresponding IPR bit is set.<br>1 Further interrupts from this source are disabled. |
| 1 | A | Activity. Indicates an interrupt has been requested or is in service. The VECTOR and PRIORITY values should not be changed while this bit is set.<br>0 No current interrupt activity associated with this source.<br>1 The interrupt field for this source is set in the IPR or ISR. |
| 2–11 | — | Reserved, should be cleared. |
| 12–15 | PRIORITY | Priority. Specifies the interrupt priority. The lowest priority is 0 and the highest priority is 15. A priority level of 0 inhibits signalling of this interrupt to the core. Affects only interrupts routed to *int*. |
| 16–31 | VECTOR | Vector (Affects only interrupts routed to *int*). Contains the value returned when IACK is read and this interrupt resides in the corresponding interrupt request register (IRR) for that core, as shown in Figure 9-50. |

### 9.3.1.8 Spurious Vector Register (SVR)

SVR, shown in Figure 9-10, contains the 16-bit vector returned to the processor core when the corresponding IACK register is read for a spurious interrupt.

Offset 0x10E0 Access: Read/Write

| | 0–15 | 16–31 |
|---|---|---|
| R<br>W | — | VECTOR |
| Reset | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 |

**Figure 9-10. Spurious Vector Register (SVR)**

Table 9-12 describes the SVR fields.

**Table 9-12. SVR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–15 | — | Reserved, should be cleared. |
| 16–31 | VECTOR | Spurious interrupt vector. Value returned when IACK is read during a spurious vector fetch. Section 9.4.1.2.3, "Spurious Vector Generation," gives information about the conditions that may cause a spurious vector fetch. |

## 9.3.2 Global Timer Registers

The two independent groups of global timer registers, group A and group B, are identical in their functionality, except that they appear at different locations within the PIC register map. Note that each of the four timers within an *x* group have four individual configuration registers (GTCCR*xn*, GTBCR*xn*, GTVPR*xn*, GTDR*xn*), but they are only shown once in this section. These two groups of timers cannot be cascaded together.

### 9.3.2.1 Timer Frequency Reporting Register (TFRRA–TFRRB)

The TFRRs, shown in Figure 9-11, are written by software to report the clocking frequency of the PIC timers. Note that although TFRRs are read/write, the PIC ignores the register values.

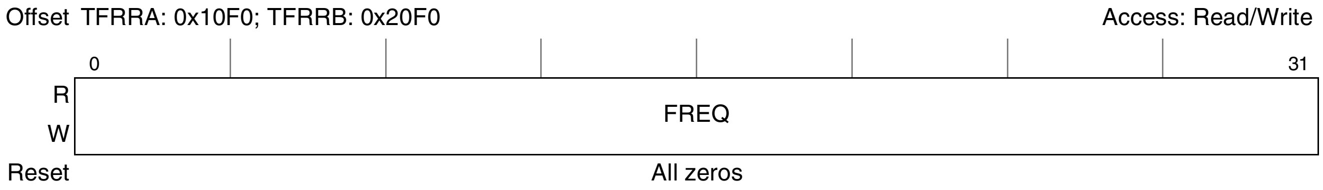


**Figure 9-11. Timer Frequency Reporting Registers (TFRR*x*)**

Table 9-13 describes the TFRR*x* registers.

**Table 9-13. TFRR*x* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–31 | FREQ | Timer frequency (in ticks/second (Hz)). Used to communicate the frequency of the global timers' clock source, (either the CCB clock or the frequency of the RTC signal), to user software. TFRR*x* is set only by software for later use by other applications and its value in no way affects the operating frequency of the global timers. The timers operate at a ratio of this clock frequency, as set by TCR*x*[CLKR]. See Section 9.3.2.6, "Timer Control Registers (TCRA–TCRB)." |

### 9.3.2.2 Global Timer Current Count Registers (GTCCRA0–GTCCRA3, GTCCRB0–GTCCRB3)

The GTCCRs, shown in Figure 9-12, contain the current count for each of the four PIC timers in each of the two groups.

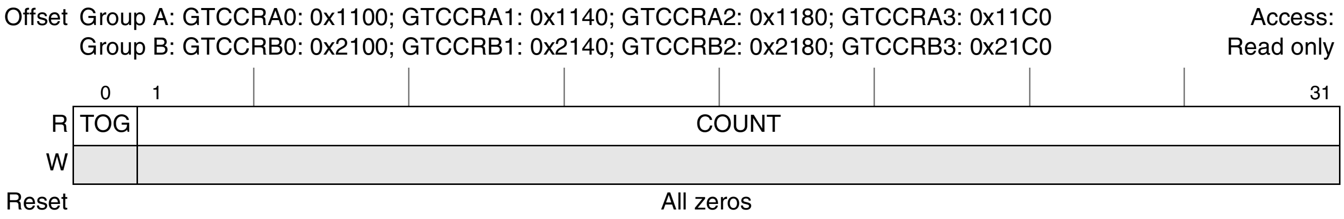

Offset Group A: GTCCRA0: 0x1100; GTCCRA1: 0x1140; GTCCRA2: 0x1180; GTCCRA3: 0x11C0
Group B: GTCCRB0: 0x2100; GTCCRB1: 0x2140; GTCCRB2: 0x2180; GTCCRB3: 0x21C0
Access: Read only
0
1
31
R
TOG
COUNT
W
Reset
All zeros

**Figure 9-12. Global Timer Current Count Registers (GTCCR*xn*)**

Table 9-14 describes the GTCCR*xn* fields.

**Table 9-14. GTCCR*xn* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | TOG | Toggle. Toggles when the current count decrements to zero. Cleared when GTBCR*xn*[CI] goes from 1 to 0. |
| 1–31 | COUNT | Current count. Decremented while GTBCR*xn*[CI] is zero. When the timer count reaches zero, an interrupt is generated (provided it is not masked), the toggle bit is inverted, and the count is reloaded. For non-cascaded timers, the reload value is the contents of the corresponding GTBCR*xn*. Cascaded timers are reloaded with either all ones, or the GTBCR*xn* contents, depending on the value of TCR*n*[ROVR]. See Section 9.3.2.6, "Timer Control Registers (TCRA–TCRB)," for more details. |

### 9.3.2.3 Global Timer Base Count Registers (GTBCRA0–GTBCRA3, GTBCRB0–GTBCRB3)

The GTBCRs contain the base counts for each of the four PIC timers in each of the two groups, as shown in Figure 9-13. This value is reloaded into the corresponding GTCCR*xn* when the current count reaches zero. Note that when zero is written to the base count field, (and GTCCR*xn*[CI] = 0), the timer generates an interrupt on every timer cycle.

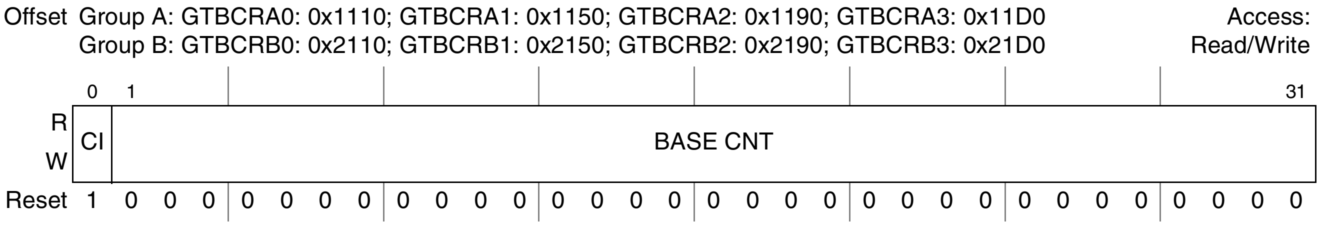

Offset Group A: GTBCRA0: 0x1110; GTBCRA1: 0x1150; GTBCRA2: 0x1190; GTBCRA3: 0x11D0
Group B: GTBCRB0: 0x2110; GTBCRB1: 0x2150; GTBCRB2: 0x2190; GTBCRB3: 0x21D0
Access: Read/Write
0
1
31
R
W
CI
BASE CNT
Reset 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

**Figure 9-13. Global Timer Base Count Register (GTBCR*xn*)**

Table 9-15 describes the GTBCR*xn* fields.

**Table 9-15. GTBCR*xn* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | CI | Count inhibit. Always set following reset<br>0 Counting enabled<br>1 Counting inhibited |
| 1–31 | BASE CNT | Base count. When CI transitions from 1 to 0, this value is copied into the corresponding GTCCR*xn* and the toggle bit is cleared. If CI is already cleared (counting is in progress), the base count is copied to the GTCCR*xn* at the next zero crossing of the current count. |

### 9.3.2.4 Global Timer Vector/Priority Registers (GTVPRA0–GTVPRA3, GTVPRB0–GTVPRB3)

The GTVPRs contain the interrupt vector and the interrupt priority values for the timers as shown in Figure 9-14. They also contain the mask and activity fields for all the timers. See Section 9.4.1, “Flow of Interrupt Control,” for information on IPR and ISR.

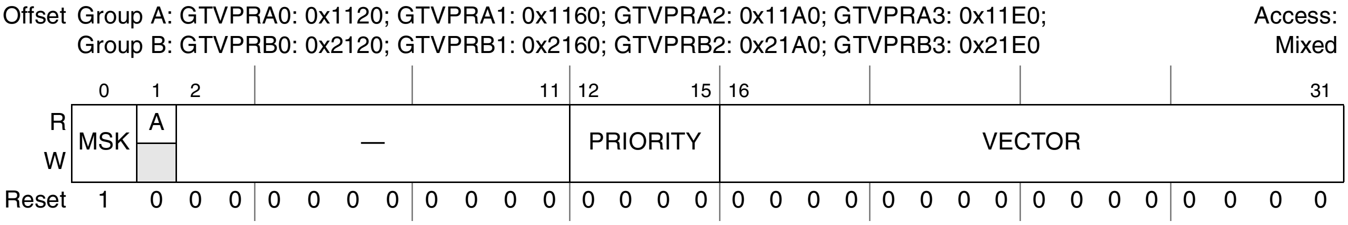

Offset Group A: GTVPRA0: 0x1120; GTVPRA1: 0x1160; GTVPRA2: 0x11A0; GTVPRA3: 0x11E0;
Group B: GTVPRB0: 0x2120; GTVPRB1: 0x2160; GTVPRB2: 0x21A0; GTVPRB3: 0x21E0
Access: Mixed
0
1
2
11
12
15
16
31
R
W
MSK
A
—
PRIORITY
VECTOR
Reset 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

**Figure 9-14. Global Timer Vector/Priority Register (GTVPR*xn*)**

Table 9-16 describes the GTVPR*xn* fields.

**Table 9-16. GTVPR*xn* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | MSK | Mask. Mask interrupts to *int* from this source.<br>0 An interrupt request is generated if the corresponding IPR bit is set.<br>1 Further interrupts from this source are disabled. |
| 1 | A | Activity. Indicates an interrupt has been requested or is in service. The VECTOR and PRIORITY values should not be changed while this bit is set.<br>0 No current interrupt activity associated with this source.<br>1 The interrupt field for this source is set in the IPR or ISR. |
| 2–11 | — | Reserved, should be cleared. |
| 12–15 | PRIORITY | Priority. Specifies the interrupt priority. The lowest priority is 0 and the highest priority is 15. A priority level of 0 inhibits signalling of this interrupt to the core. Affects only interrupts routed to *int*. |
| 16–31 | VECTOR | Vector (Affects only interrupts routed to *int*). Contains the value returned when IACK is read and this interrupt resides in the corresponding interrupt request register (IRR) for that core, as shown in Figure 9-50. |

### 9.3.2.5 Global Timer Destination Registers (GTDRA0–GTDRA3, GTDRB0–GTDRB3)

The GTDR*xn* registers, shown in Figure 9-15, control the destination (core) to which each timer's interrupt is directed. Note that GTDR*xn* bits can be set independently of each other and that either P1 or P0 or both can be set for this type of interrupt.

Offset Group A: GDTRA0: 0x1130; GDTRA1: 0x1170; GDTRA2: 0x11B0; GDTRA3: 0x11F0;
Group B: GDTRB0: 0x2130; GDTRB1: 0x2170; GDTRB2: 0x21B0; GDTRB3: 0x21F0

Access: Read/Write

|  | 0–29 | 30 | 31 |
|---|---|---|---|
| R<br>W | — | P1[1] | P0 |
| Reset | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 | 1 |

[1] Reserved in single-processor implementations.

**Figure 9-15. Global Timer Destination Registers (GTDR*xn*)**

Table 9-17 describes the GTDR*xn* fields.

**Table 9-17. GTDR*xn* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–29 | — | Reserved, should be cleared. |
| 30 | P1 | Processor core 1. This interrupt is multicasting, so both P0 and P1 can be set.<br>0 Processor core 1 does not receive this interrupt<br>1 Directs the timer interrupt to processor core 1<br>**Note:** Reserved in single-processor implementations. |
| 31 | P0 | Processor core 0. Default destination after PIC is reset. Both P0 and P1 can be set.<br>0 Processor core 0 does not receive this interrupt.<br>1 Directs the timer interrupt to processor core 0. |

### 9.3.2.6 Timer Control Registers (TCRA–TCRB)

The TCR registers, shown in Figure 9-17, provide various configuration options such as count frequency and roll-over behavior for the timers.

There are two choices for the clock source for the timers: a selectable frequency ratio from the CCB bus clock, or the RTC signal. TCRs can be cascaded to create timers larger than the default 31-bit global timers. Timer cascade fields allow configuration of up to two 63-bit timers, one 95-bit timer, or one 127-bit timer (within each group).

With one exception mentioned below, the value reloaded into a timer is determined by its roll-over control field, TCR*x*[ROVR]. Setting TCR*x*[ROVR] causes its GTCCR*xn* to roll over to all ones when the count reaches zero. This is equivalent to reloading the count register with 0xFFFF_FFFF instead of its base count value. Clearing a timer's associated ROVR bit ensures the timer always reloads with its base count value.

When timers are cascaded, the last (most significant) counter in the cascade also affects their roll-over behavior. Cascaded timers always reload their base count when the most significant counter has decremented to zero, regardless of the TCR*x*[ROVR] settings.

For example, timers 0–2 can be cascaded to generate one interrupt per hour. As shown in Table 9-18, given an CCB clock frequency of 333 MHz, letting the timer clock frequency default to 1/8th the system clock, (TCR*x*[CLKR] = 0 sets a clock ratio of 8), provides a basic input of 41.625 MHz to timer 0. Setting timer 0 to count 41,625,000 (0x27B_25A8) timer clock cycles generates one output per second. Setting both timers 1 and 2 to 59, and cascading all three timers, generates one interrupt every hour from timer 2.

**Table 9-18. Parameters for Hourly Interrupt Timer Cascade Example**

| System Clock | Clock Ratio | Timer Clock | Timer 0 Count | Timer 1 Count | Timer 2 Count |
|---|---|---|---|---|---|
| 333 MHz | 1 / 8 | 41.625 MHz | $41.625 \times 10^6$<br>(0x027B_25A8) | 59[1]<br>(0x0000_0036) | 59<br>(0x0000_0036) |

[1] Counting down from 59 through 0 requires 60 ticks.

(41.625 x 106 ticks/sec) × (60 sec/min) × (60 min/hr) = total ticks/hr generating 1 interrupt/hr

**Figure 9-16. Example Calculation for Cascaded Timers**

Offset TCRA: 0x1300; TCRB: 0x2300 Access: Read/Write

| | 0–4 | 5–7 | 8–14 | 15 | 16–21 | 22–23 | 24–28 | 29–31 |
|---|---|---|---|---|---|---|---|---|
| R<br>W | — | ROVR | — | RTM | — | CLKR | — | CASC |
| Reset | All zeros | | | | | | | |

**Figure 9-17. Timer Control Registers (TCR*x*)**

Table 9-19 describes the TCR*x* fields.

**Table 9-19. TCR*x* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–4 | — | Reserved, should be cleared. |
| 5–7 | ROVR | Roll-over control for cascaded timers only. Specifies behavior when count reaches zero by identifying the source of the reload value. Cascaded timers are always reloaded with their base count value when the more significant timer in the cascade (the upstream timer) is zero. Bits 5–7 correspond to timers 2–0. Note that global timer 3 always reloads with its GTBCR*xn*.<br>0 The timer does not roll over. When the count reaches zero, GTCCR*xn* is reloaded with the GTBCR*xn* value.<br>1 Timer rolls over at zero to all ones. (When the count reaches zero, GTCCR*xn* is reloaded with 0xFFFF_FFFF.)<br>000 All timers reload with base count.<br>001 Timers 1 and 2 reload with base count, timer 0 rolls over (reloads with 0xFFFF_FFFF).<br>010 Timers 0 and 2 reload with base count, timer 1 rolls over (reloads with 0xFFFF_FFFF).<br>011 Timer 2 reloads with base count, timers 0 and 1 roll over (reload with 0xFFFF_FFFF).<br>100 Timers 0 and 1 reload with base count, timer 2 rolls over (reloads with 0xFFFF_FFFF).<br>101 Timer 1 reloads with base count, timers 0 and 2 roll over (reload with 0xFFFF_FFFF).<br>110 Timer 0 reloads with base count, timers 1 and 2 roll over (reload with 0xFFFF_FFFF).<br>111 Timers 0, 1, and 2 roll over (reload with 0xFFFF_FFFF). |
| 8–14 | — | Reserved, should be cleared. |
| 15 | RTM | Real time mode. Specifies the clock source for the PIC timers.<br>0 Timer clock frequency is a ratio of the frequency of the CCB clock as determined by the CLKR field. This is the default value.<br>1 The RTC signal is used to clock the PIC timers. If this bit is set, the CLKR field has no meaning. |
| 16–21 | — | Reserved, should be cleared. |
| 22–23 | CLKR | Clock ratio. Specifies the ratio of the timer frequency to the CCB clock. The following are supported:<br>00 Default. Divide by 8<br>01 Divide by 16<br>10 Divide by 32<br>11 Divide by 64 |
| 24–28 | — | Reserved, should be cleared. |
| 29–31 | CASC | Cascade timers. Specifies the output of particular global timers as input to others.<br>000 Default. Timers not cascaded<br>001 Cascade timers 0 and 1<br>010 Cascade timers 1 and 2<br>011 Cascade timers 0, 1, and 2<br>100 Cascade timers 2 and 3<br>101 Cascade timers 0 and 1; timers 2 and 3<br>110 Cascade timers 1, 2, and 3<br>111 Cascade timers 0, 1, 2, and 3 |

### 9.3.3 IRQ_OUT and Critical Interrupt Summary Registers

The summary registers indicate the specific interrupt sources routed to the $\overline{\text{IRQ\_OUT}}$ or *cint0/cint1*. PIC outputs. Summary register bits are cleared when the corresponding interrupt that caused a bit to be set is negated. Note that only level-sensitive interrupts can be routed to $\overline{\text{IRQ\_OUT}}$ or *cint0* and *cint1*.

The IRQ_OUT summary registers, shown in Figure 9-19 through Figure 9-21 contain one bit for each interrupt source that can be routed to $\overline{\text{IRQ\_OUT}}$. The corresponding bit is set if the interrupt is active and is routed to $\overline{\text{IRQ\_OUT}}$ (that is, if the corresponding *x*IDR*n*[EP] is set).

The critical interrupt summary registers, shown in Figure 9-22 through Figure 9-24, contain one bit for each interrupt source that can be designated as a critical interrupt. The corresponding bit is set if the interrupt is active and is routed to either the *cint* outputs of the PIC (if *x*IDR*n*[CI*n*] = 1 in its corresponding destination register).

### 9.3.3.1 External Interrupt Summary Register (ERQSR)

**NOTE**

> ERQSR fields report only the current state of IRQ0–IRQ11 pins. These fields were designed to work with level-sensitive interrupts; values returned for edge-sensitive interrupts may be unreliable.

Figure 9-18 shows the ERQSR fields.

Offset 0x1308 Access: Read only

| | 0 | 6 | 7 | 8 | 11 | 12 | | | | 31 |
|---|---|---|---|---|---|---|---|---|---|---|
| Reset | All zeros | | | | | | | | | |

**Figure 9-18. External Interrupt Summary Register (ERQSR)**

Table 9-20 describes the ERQSR fields.

**Table 9-20. ERQSR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| | EINT*n* | External interrupts signal status. Bit 0 represents EINT0. Bit 11 represents EINT11.<br>0 The corresponding external interrupt signal is not active.<br>1 The corresponding external interrupt signal is active. |
| 12–31 | — | Reserved, should be cleared. |

### 9.3.3.2 IRQ_OUT Summary Register 0 (IRQSR0)

Figure 9-19 shows the IRQSR0 fields.

Offset 0x1310 Access: Read only

| | 0–7 | 8–11 | 12–15 | 16–19 | 20–31 |
|---|---|---|---|---|---|
| R | MSI0–MSI7 | — | MSG4–MSG7 | MSG0–MSG3 | EXT*n* |
| W | | | | | |
| Reset | All zeros | | | | |

**Figure 9-19. IRQ_OUT Summary Register 0 (IRQSR0)**

Table 9-21 describes the IRQSR0 fields.

**Table 9-21. IRQSR0 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–7 | MSI*n* | Shared message signaled interrupt *n* status<br>0 Interrupt is not active or not routed to $\overline{\text{IRQ\_OUT}}$.<br>1 Interrupt is active and is routed to the $\overline{\text{IRQ\_OUT}}$ signal (that is, if the corresponding *x*IDR*n*[EP] is set). |
| 8–11 | — | Reserved, should be cleared. |
| 12–15 | MSG*n* | Message interrupt *n* status<br>0 Interrupt is not active or not routed to $\overline{\text{IRQ\_OUT}}$.<br>1 Interrupt is active and is routed to the $\overline{\text{IRQ\_OUT}}$ signal (that is, if the corresponding *x*IDR*n*[EP] is set). |
| 16–19 | MSG*n* | Message interrupt *n* status<br>0 Interrupt is not active or not routed to $\overline{\text{IRQ\_OUT}}$.<br>1 Interrupt is active and is routed to the $\overline{\text{IRQ\_OUT}}$ signal (that is, if the corresponding *x*IDR*n*[EP] is set). |
| 20–31 | EXT*n* | External interrupts . Each bit corresponds to a unique interrupt according to the following:<br>Bit Interrupt<br>20 IRQ0<br>21 IRQ1<br>…<br>31 IRQ11<br>0 The corresponding interrupt is not active or not routed to $\overline{\text{IRQ\_OUT}}$.<br>1 The corresponding interrupt is active and routed to $\overline{\text{IRQ\_OUT}}$ (if the corresponding *x*IDR*n*[EP] is set). |

## 9.3.3.3 IRQ_OUT Summary Register 1 (IRQSR1)

Figure 9-20 shows the IRQSR1 fields.

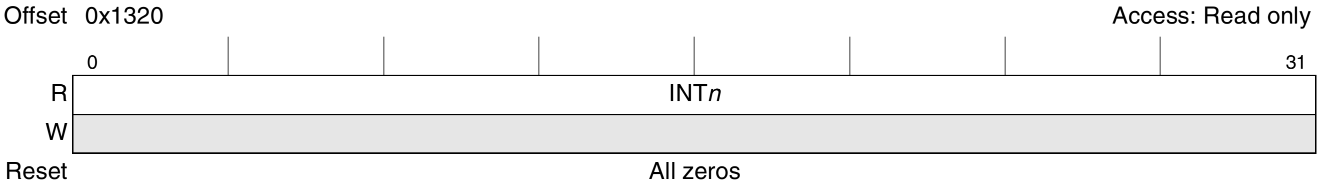


**Figure 9-20. IRQ_OUT Summary Register 1 (IRQSR1)**

Table 9-22 describes the IRQSR1 fields.

**Table 9-22. IRQSR1 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–31 | INT*n* | Internal interrupts 0–31 status. Bit 0 represents INT0. Bit 31 represents INT31.<br>0 The corresponding interrupt is not active or not routed to $\overline{\text{IRQ\_OUT}}$.<br>1 The corresponding interrupt is active and is routed to $\overline{\text{IRQ\_OUT}}$ (that is, if the corresponding *x*IDR*n*[EP] is set). |

### 9.3.3.4 IRQ_OUT Summary Register 2 (IRQSR2)

Figure 9-21 shows the IRQSR2 fields.

Offset 0x1324 Access: Read only

| | 0 — 31 |
|---|---|
| R | IINT*n* |
| W | |
| Reset | All zeros |

**Figure 9-21. IRQ_OUT Summary Register 2 (IRQSR2)**

Table 9-23 describes the IRQSR2 fields.

**Table 9-23. IRQSR2 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–31 | INT*n* | Internal interrupts 32–63 status. Bit 0 represents INT32. Bit 31 represents INT63.<br>0 The corresponding interrupt is not active or not routed to $\overline{\text{IRQ\_OUT}}$.<br>1 The corresponding interrupt is active and is routed to $\overline{\text{IRQ\_OUT}}$, if the corresponding *x*IDR*n*[EP] is set. |

### 9.3.3.5 Critical Interrupt Summary Register 0 (CISR0)

Figure 9-22 shows CISR0.

Offset 0x1330 Access: Read only

| | 0 — 7 | 8 — 11 | 12 — 15 | 16 — 19 | 20 — 31 |
|---|---|---|---|---|---|
| R | MSI0–MSI7 | — | MSG4-MSG7 | MSG0–MSG3 | EXT*n* |
| W | | | | | |
| Reset | All zeros | | | | |

**Figure 9-22. Critical Interrupt Summary Register 0 (CISR0)**

Table 9-24 describes CISR0 fields.

**Table 9-24. CISR0 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–7 | MSI*n* | Shared message signaled interrupts 0–7. Bit 0 represents MSI0; bit 7 represents MSI7.<br>0 The corresponding interrupt is not active or not routed to *cint*.<br>1 The corresponding interrupt is active and is routed to *cint*, if the corresponding *x*IDR*n*[CI] is set. |
| 8–11 | — | Reserved, should be cleared. |
| 12–15 | MSG*n* | Message interrupts 4–7. Bit 16 represents MSG4; bit 19 represents MSG7.<br>0 The corresponding interrupt is not active or not routed to *cint*.<br>1 The corresponding interrupt is active and is routed to *cint* (if the corresponding *x*IDR*n*[CI] is set). |
| 16–19 | MSG*n* | Message interrupts 0–3. Bit 16 represents MSG0; bit 19 represents MSG3.<br>0 The corresponding interrupt is not active or not routed to *cint*.<br>1 The corresponding interrupt is active and is routed to *cint* (if the corresponding *x*IDR*n*[CI] is set). |
| 20–31 | EXT*n* | External interrupts . Bit 20 represents IRQ0. Bit 31 represents IRQ11.<br>0 The corresponding interrupt is not active or not routed to *cint*.<br>1 The corresponding interrupt is active and is routed to *cint* (if the corresponding *x*IDR*n*[CI] is set). |

### 9.3.3.6 Critical Interrupt Summary Register 1 (CISR1)

Figure 9-23 shows the CISR1.

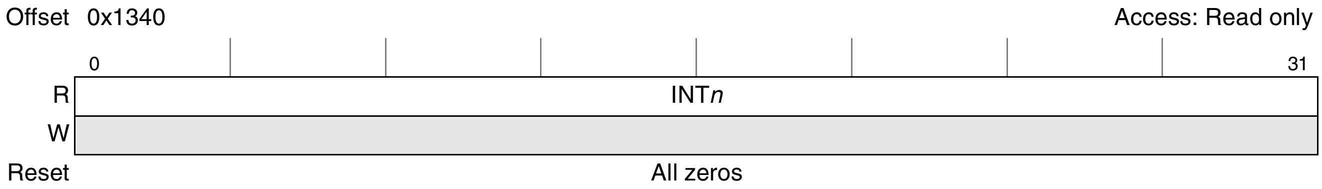


**Figure 9-23. Critical Interrupt Summary Register 1 (CISR1)**

Table 9-25 describes CISR1.

**Table 9-25. CISR1 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–31 | INT*n* | Internal interrupts 0–31. Bit 0 represents INT0. Bit 31 represents INT31.<br>0 Corresponding interrupt is not active or not routed to *cint*.<br>1 The corresponding interrupt is active and is routed to the *cint* (if the corresponding *x*IDR*n*[CI] is set). |

### 9.3.3.7 Critical Interrupt Summary Register 2 (CISR2)

Figure 9-24 shows the CISR2.

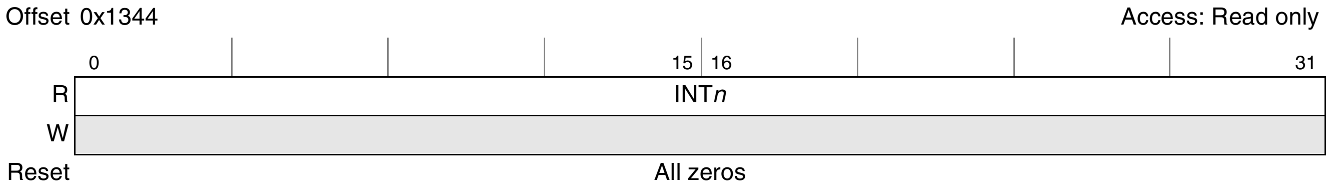


**Figure 9-24. Critical Interrupt Summary Register 2 (CISR2)**

Table 9-26 describes CISR2.

**Table 9-26. CISR2 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–31 | INT*n* | Internal interrupts 32–63. Bit 0 represents INT32. Bit 31 represents INT63.<br>0 Corresponding interrupt is not active or not routed to *cint*.<br>1 The corresponding interrupt is active and is routed to the *cint*, if the corresponding *x*IDR*n*[CI] is set. |

## 9.3.4 Performance Monitor Mask Registers (PMMRs)

The twelve performance monitor mask registers consist of four sets of three 32-bit registers, PM*n*MR0, PM*n*MR1, and PM*n*MR2. Each set can be configured to select one interrupt source (interprocessor, timer, message, shared message signaled, external, or internal) to generate a performance monitor event. The performance monitor can be configured to track this event in the performance monitor local control registers. See Section 24.3.2.2, "Performance Monitor Local Control Registers (PMLCAn, PMLCBn)."

### 9.3.4.1 Performance Monitor Mask Registers 0 (PM0MR0–PM3MR0)

Each PM*n*MR0 register, shown in Figure 9-25, is matched with a PM*n*MR1 and a PM*n*MR2 register. Because each unreserved bit in the 96-bit vector (PM*n*MR0/1/2) specifies a different interrupt, only one bit in the 96-bit vector can be unmasked at a time. Unmasking more than one bit per set is considered a programming error and results in unpredictable behavior.

Offset PM0MR0: 0x1350; PM1MR0: 0x1370; PM2MR0: 0x1390; PM3MR0: 0x13B0 Access: Read/Write

|  | 0 … 7 | 8 … 11 | 12 … 15 | 16 … 19 | 20 … 31 |
|---|---|---|---|---|---|
| R<br>W | MShI | IPI | TIMER | MSG | EXT |
| Reset | 1 1 1 1 1 1 1 1 | 1 1 1 1 | 1 1 1 1 | 1 1 1 1 | 1 1 1 1 1 1 1 1 1 1 1 1 |

**Figure 9-25. Performance Monitor Mask Registers 0 (PM*n*MR0)**

Table 9-27 describes the PM*n*MR0 fields.

**Table 9-27. PM*n*MR0 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–7 | MShI | Shared message signaled interrupts 0–7<br>0 The corresponding interrupt source generates a performance monitor event when the interrupt occurs.<br>1 The corresponding interrupt does not generate a performance monitor event. |
| 8–11 | IPI | Interprocessor interrupts 0–3<br>0 The corresponding interrupt source generates a performance monitor event when the interrupt occurs.<br>1 The corresponding interrupt does not generate a performance monitor event. |
| 12–15 | TIMER | Timer interrupts 0–3 (Group A and Group B: Each bit represents an OR of the event for the correspondingly numbered timer in Group A and that in Group B).<br>0 The corresponding interrupt source generates a performance monitor event when the interrupt occurs.<br>1 The corresponding interrupt does not generate a performance monitor event. |
| 16–19 | MSG | Message interrupts 0–7<br>Bit 0 is used for MSG0 and MSG4<br>Bit 1 is used for MSG1 and MSG5<br>Bit 2 is used for MSG2 and MSG6<br>Bit 3 is used for MSG3 and MSG7<br>0 The corresponding interrupt source generates a performance monitor event when the interrupt occurs.<br>1 The corresponding interrupt does not generate a performance monitor event. |
| 20–31 | EXT | External interrupts IRQ[]<br>0 The corresponding interrupt source generates a performance monitor event when the interrupt occurs.<br>1 The corresponding interrupt does not generate a performance monitor event. |

### 9.3.4.2 Performance Monitor Mask Registers 1 (PM0MR1–PM3MR1)

Figure 9-26 shows the PM*n*MR1 registers.

Offset PM0MR1: 0x1360; PM1MR1: 0x1380; PM2MR1: 0x13A0; PM3MR1: 0x13C0 Access: Read/Write

| | 0 … 31 |
|---|---|
| R<br>W | INT |
| Reset | All ones |

**Figure 9-26. Performance Monitor Mask Registers 1 (PM*n*MR1)**

Table 9-28 describes the PM*n*MR1 registers.

**Table 9-28. PM*n*MR1 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–31 | INT | Internal interrupts 0–31<br>0 The corresponding interrupt source generates a performance monitor event when the interrupt occurs.<br>1 The corresponding interrupt does not generate a performance monitor event. |

### 9.3.4.3 Performance Monitor Mask Registers 2 (PM0MR2–PM3MR2)

Figure 9-27 shows the PM*n*MR2 registers.

Offset PM0MR2: 0x1364; PM1MR2: 0x1384; PM2MR2: 0x13A4; PM3MR2: 0x13C4 Access: Read/Write

| | 0 … 31 |
|---|---|
| R<br>W | INT |
| Reset | 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 |

**Figure 9-27. Performance Monitor Mask Registers 2 (PM*n*MR2)**

Table 9-29 describes the PM*n*MR2 registers.

**Table 9-29. PM*n*MR2 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–31 | INT | Internal interrupts 32–64<br>0 The corresponding interrupt source generates a performance monitor event when the interrupt occurs.<br>1 The corresponding interrupt does not generate a performance monitor event. |

## 9.3.5 Message Registers

The following registers support the message register interrupts:

- Section 9.3.5.1, “Message Registers (MSGR0–MSGR7)”
- Section 9.3.5.2, “Message Enable Register (MER)”
- Section 9.3.5.3, “Message Status Register (MSR)”
- Section 9.3.7.5, “Messaging Interrupt Vector/Priority Registers (MIVPRn)”

- Section 9.3.7.6, “Messaging Interrupt Destination Registers (MIDR0–MIDR7)”

Writing to one of the four message registers (MSGR0–MSGR7) causes a messaging interrupt as directed by the other message registers listed above. Reading a message register clears the messaging interrupt. Note that a messaging interrupt can also be cleared by writing a one to the corresponding status field of the PIC message status register (MSR), shown in Figure 9-30.

### 9.3.5.1 Message Registers (MSGR0–MSGR7)

The message registers (MSGR0–MSGR7), shown in Figure 9-28, can contain a 32-bit message.

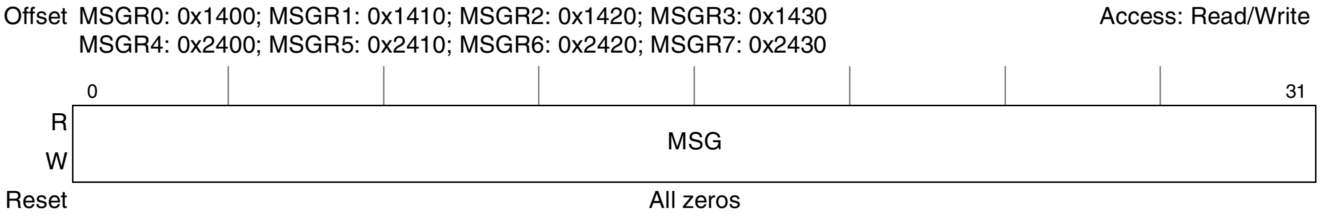

Offset MSGR0: 0x1400; MSGR1: 0x1410; MSGR2: 0x1420; MSGR3: 0x1430
MSGR4: 0x2400; MSGR5: 0x2410; MSGR6: 0x2420; MSGR7: 0x2430
Access: Read/Write
0
31
R
W
MSG
Reset
All zeros

**Figure 9-28. Message Registers (MSGRs)**

Table 9-30 describes the MSGR registers.

**Table 9-30. MSGR*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–31 | MSG | Message. Contains the 32-bit message data. |

### 9.3.5.2 Message Enable Register (MER)

The MER, shown in Figure 9-29, contains the enable bits for each message register. The enable bit must be set to enable interrupt generation when the corresponding message register is written.

When bits in MER are set to mask message interrupts, an interrupt is not generated if the message register is written while it is masked in MER and the MER bit is then cleared. To mask the interrupt without loss, set MIVPR*n*[MSK]. (See Section 9.3.7.5, “Messaging Interrupt Vector/Priority Registers (MIVPRn).”) MER should be set to 0x0000_000F at reset and then left unchanged during normal operation.

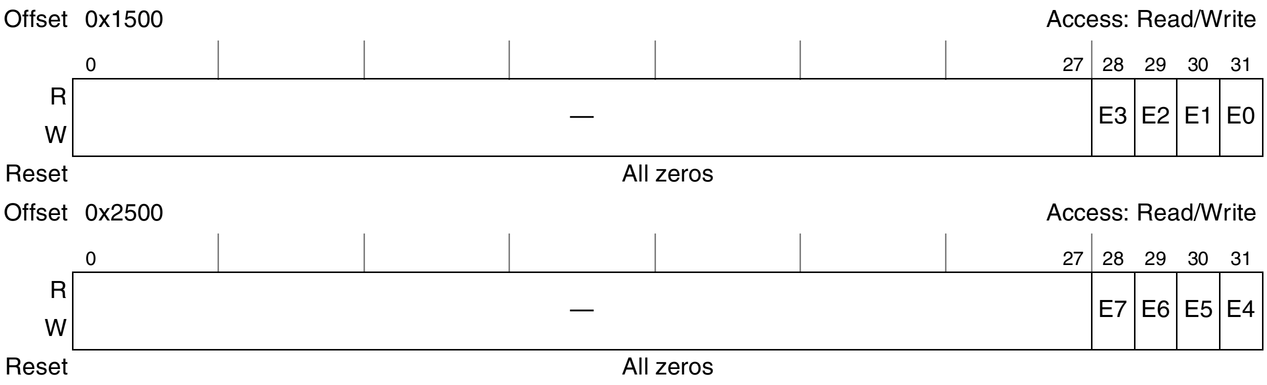


**Figure 9-29. Message Enable Register (MER)**

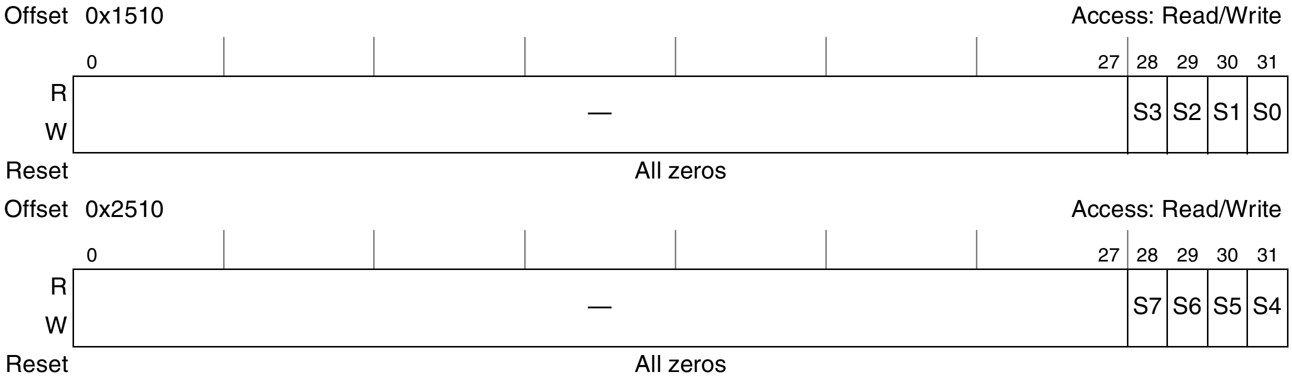
Table 9-31 describes the MER fields.

**Table 9-31. MER Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–27 | — | Reserved, should be cleared. |
| 28–32 | E*n* | Enable 3–enable 0 or enable 7–enable 0.<br>Used to enable interrupt generation for MSGR*n* (where *n* = 0–7).<br>0 Interrupt generation for MSGR*n* disabled.<br>1 Interrupt generation for MSGR*n* enabled. |

### 9.3.5.3 Message Status Register (MSR)

The message status register (MSR) shown in Figure 9-30 contains status bits for each message register. A status bit is set when the corresponding messaging interrupt is active. Writing a 1 to a status bit clears the corresponding message interrupt and the status bit.



Offset 0x1510 Access: Read/Write

| | 0–27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|
| R<br>W | — | S3 | S2 | S1 | S0 |
| Reset | All zeros | | | | |

Offset 0x2510 Access: Read/Write

| | 0–27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|
| R<br>W | — | S7 | S6 | S5 | S4 |
| Reset | All zeros | | | | |

**Figure 9-30. Message Status Register (MSR)**

Table 9-32 describes the MSR fields.

**Table 9-32. MSR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–27 | — | Reserved, should be cleared. |
| 28 | S*n* | Status 3–status 0 or status 7–status 4. Reports status of messaging interrupt *n*. Writing a 1 clears this field.<br>0 Messaging interrupt *n* is not active.<br>1 Messaging interrupt *n* is active. |

## 9.3.6 Shared Message Signaled Registers

This section contains description the shared message signaled interrupt registers (MSIRs). The shared message signaled interrupt structure allows programs to interrupt each other by simply writing to these shared memory-mapped registers in the PIC. Each of the eight MSIRs can be thought of as collecting interrupts from 32 different memory-mapped writes that can cause interrupts.

### 9.3.6.1 Shared Message Signaled Interrupt Registers (MSIR0–MSIR7)

The eight MSIRs indicate which of the up to 32 interrupt sources sharing the message register have pending interrupts. These registers are cleared when read. A write to these registers has no effect.

Offset MSIR0: 0x1600; MSIR1: 0x1610; MSIR2: 0x1620; MSIR3: 0x1630; MSIR4: 0x1640; MSIR5: 0x1650; MSIR6: 0x1660; MSIR7: 0x1670

Access: Read only

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | SH31 | SH30 | SH29 | SH28 | SH27 | SH26 | SH25 | SH24 | SH23 | SH22 | SH21 | SH20 | SH19 | SH18 | SH17 | SH16 |
| W | | | | | | | | | | | | | | | | |
| Reset | All zeros | | | | | | | | | | | | | | | |

| | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | SH15 | SH14 | SH13 | SH12 | SH11 | SH10 | SH9 | SH8 | SH7 | SH6 | SH5 | SH4 | SH3 | SH2 | SH1 | SH0 |
| W | | | | | | | | | | | | | | | | |
| Reset | All zeros (Note: After soft reset, read the MSIRs to ensure that any interrupts previously pending are cleared). | | | | | | | | | | | | | | | |

**Figure 9-31. Message Signaled Interrupt Registers (MSIR*n*)**

Table 9-33 describes the bits of the MSIRs.

**Table 9-33. MSIR*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| *n* | SH*n* | Message sharer *n* has a pending interrupt. |

### 9.3.6.2 Shared Message Signaled Interrupt Status Register (MSISR)

MSISR, shown in Figure 9-32, contains the status bits for the shared message signaled interrupts. A status bit is set when the corresponding MSIR has an active interrupt. The status bit is 0 if all the corresponding shared interrupt sources are cleared for that MSIR.

Offset 0x1720

Access: Read only

| | 0–23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|
| R | — | S7 | S6 | S5 | S4 | S3 | S2 | S1 | S0 |
| W | | | | | | | | | |
| Reset | All zeros | | | | | | | | |

**Figure 9-32. Shared Message Signaled Interrupt Status Register (MSISR)**

Table 9-34 describes the bits of the MSISR.

**Table 9-34. MSISR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–23 | — | Reserved, should be cleared. |
| 24–31 | S*n* | Status *n*.<br>0 MSIR*n* is not active.<br>1 MSIR*n* has an active interrupt. |

### 9.3.6.3 Shared Message Signaled Interrupt Index Register (MSIIR)

MSIIR, shown in Figure 9-32, provides the mechanism for setting an interrupt in the MSIRs. When MSIIR is written, MSIIR[SRS] selects the register in which an interrupt bit is to be set; MSIIR[IBS] selects the shared interrupt field in the selected MSIR register to be set. MSIIR is primarily intended to support PCI Express MSIs.

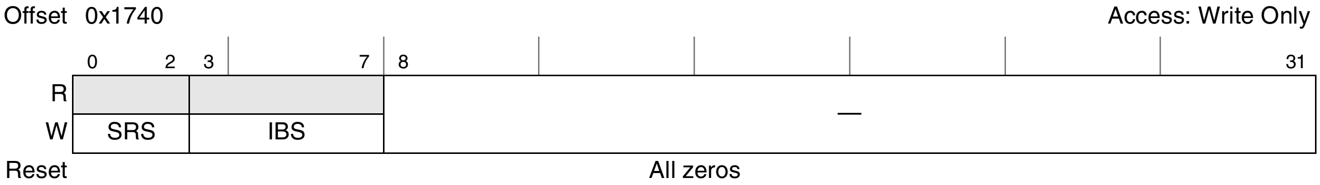


**Figure 9-33. Shared Message Signaled Interrupt Index Register (MSIIR)**

Table 9-35 describes the bits of the MSIIR.

**Table 9-35. MSIIR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–2 | SRS | Shared interrupt register select. Selects the MSIR to be written<br>000 MSIR 0<br>001 MSIR 1<br>010 MSIR 2<br>...<br>111 MSIR 7 |
| 3–7 | IBS | Interrupt bit select—Selects the bit to set in the MSIR<br>00000 Set field SH0 (bit 31)<br>00001 Set field SH1 (bit 30)<br>00010 Set field SH2 (bit 29)<br>...<br>11111 Set field SH31 (bit 0) |
| 8–31 | — | Reserved, should be cleared. |

### 9.3.6.4 Shared Message Signaled Interrupt Vector/Priority Register (MSIVPRs)

The MSIVPRs, shown in Figure 9-34, have the same fields and format as the GTVPRs. See Section 9.4.1, “Flow of Interrupt Control,” for information on IPR and ISR.

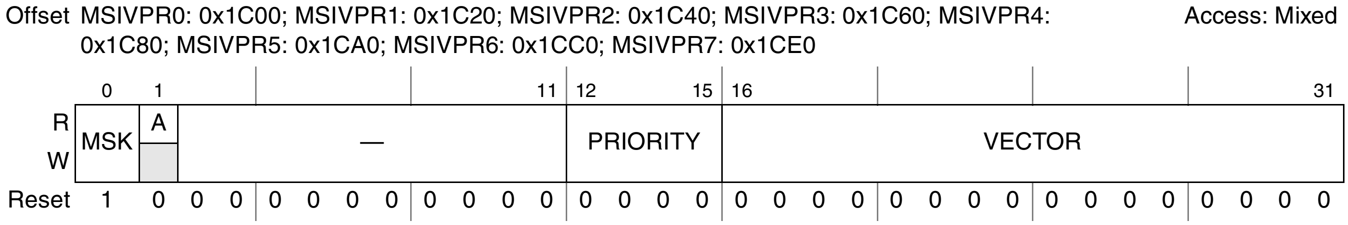

Offset MSIVPR0: 0x1C00; MSIVPR1: 0x1C20; MSIVPR2: 0x1C40; MSIVPR3: 0x1C60; MSIVPR4: 0x1C80; MSIVPR5: 0x1CA0; MSIVPR6: 0x1CC0; MSIVPR7: 0x1CE0
Access: Mixed
0
1
11
12
15
16
31
R
W
MSK
A
—
PRIORITY
VECTOR
Reset
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

**Figure 9-34. Shared Message Signaled Interrupt Vector/Priority Register (MSIVPRs)**

Table 9-36 describes the bits of the MSIVPRs.

Table 9-36. MSIVPRn Field Descriptions

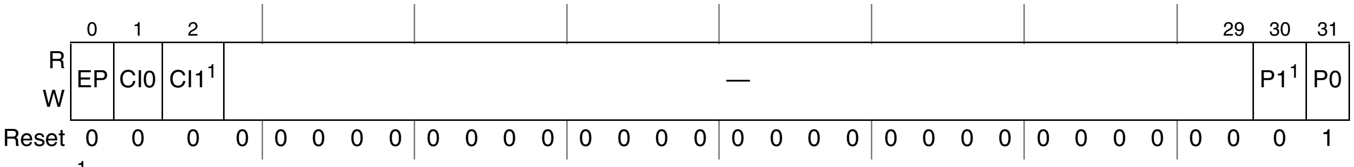
| Bits | Name | Description |
|---|---|---|
| 0 | MSK | Mask. Mask interrupts from this source. MSK affects only interrupts routed to *int*.<br>0 An interrupt request is generated if the corresponding IPR bit is set.<br>1 Further interrupts from this source are disabled. |
| 1 | A | Activity. Indicates an interrupt has been requested or is in service. The VECTOR and PRIORITY values should not be changed while this bit is set. Affects only interrupts routed to *int*.<br>0 No current interrupt activity associated with this source.<br>1 The interrupt field for this source is set in the IPR or ISR. |
| 2–11 | — | Reserved, should be cleared. |
| 12–15 | PRIORITY | Priority. Specifies the interrupt priority. The lowest priority is 0 and the highest priority is 15. A priority level of 0 inhibits signalling of this interrupt to the core. Affects only interrupts routed to *int*. |
| 16–31 | VECTOR | Vector (Affects only interrupts routed to *int*). Contains the value returned when IACK is read and this interrupt resides in the corresponding interrupt request register (IRR) for that core, as shown in Figure 9-49. |

### 9.3.6.5 Shared Message Signaled Interrupt Destination Registers 0–7 (MSIDR*n*)

The MSIDRs, shown in Figure 9-35, contain the destination fields for shared message signaled interrupts. Only one destination bit may be set; otherwise, behavior is undefined.

Offset MSIDR0: 0x1C10; MSIDR1: 0x1C30; MSIDR2: 0x1C50; MSIDR3: 0x1C70; MSIDR4: 0x1C90; MSIDR5: 0x1CB0; MSIDR6: 0x1CD0; MSIDR7: 0x1CF0

Access: Read/Write



[1] Reserved in single-processor implementations.

Figure 9-35. Shared Message Signaled Interrupt Destination Registers (MSIDR*n*)

Table 9-37 describes MSIDR*n* fields.

Table 9-37. MSIDR*n* Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0 | EP | External signal. Allows interrupt to be serviced externally.<br>0 Interrupt is not routed to $\overline{\text{IRQ\_OUT}}$.<br>1 Interrupt is routed to $\overline{\text{IRQ\_OUT}}$ for external servicing. |
| 1 | CI0 | Critical interrupt 0.<br>0 Processor core 0 does not receive this interrupt.<br>1 Directs the shared message signaled interrupt to processor core 0 by causing the *cint0* output signal from the PIC to assert. See Section 9.1.2, “Interrupts to the Processor Core.” |
| 2 | CI1 | Critical interrupt 1. Reserved in single-processor implementations.<br>0 Processor core 1 does not receive this interrupt.<br>1 Directs the shared message signaled interrupt to processor core 1 by causing the *cint1* output signal from the PIC to assert. See Section 9.1.2, “Interrupts to the Processor Core.” |

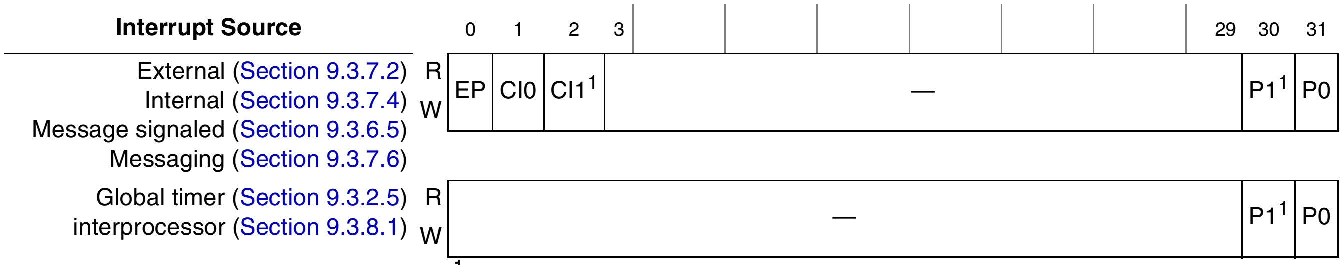
Table 9-37. MSIDR*n* Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 3–29 | — | Reserved, should be cleared. |
| 30 | P1 | Processor core 1. Indicates whether processor core 1 receives the interrupt through *int*.<br>0 Processor core 1 does not receive this interrupt.<br>1 Directs the interrupt to processor core 1 through the assertion of *int1*.<br>**Note:** Reserved in single-processor implementations. |
| 31 | P0 | Processor core 0. Indicates whether processor core 0 receives the interrupt.<br>0 Processor core 0 does not receive this interrupt.<br>1 Directs the interrupt to processor core 0 through the assertion of *int0*.<br>The default destination is for processor core 0 to receive this shared message signaled interrupt after the PIC is reset. |

### 9.3.7 Interrupt Source Configuration Registers

The interrupt source configuration registers control the source and destinations of interrupts, specifying parameters such as the interrupting event, signal polarity, and relative priority.

Figure 9-36 shows the destination registers.


Interrupt Source
0
1
2
3
29
30
31
External (Section 9.3.7.2)
Internal (Section 9.3.7.4)
Message signaled (Section 9.3.6.5)
Messaging (Section 9.3.7.6)
R
W
EP
CI0
CI1¹
—
P1¹
P0
Global timer (Section 9.3.2.5)
interprocessor (Section 9.3.8.1)
R
W
—
P1¹
P0

[1] Reserved in single-processor implementations.

Figure 9-36. Destination Register Summary

Note the following:

- The global timer and interprocessor destination register support only the P0 and P1 options. That is, they cannot be routed to *cint* or to $\overline{\text{IRQ\_OUT}}$.
- Only the global timer and interprocessor interrupts are multicasting, so only these interrupts allow more than one destination bit to be specified.

Figure 9-36 shows the vector/priority registers.

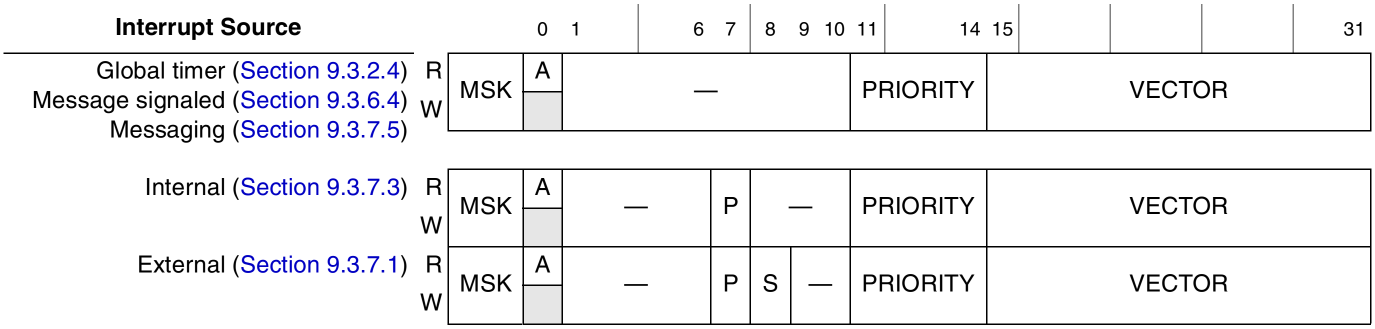

Interrupt Source
0
1
6
7
8
9
10
11
14
15
31
Global timer (Section 9.3.2.4)
Message signaled (Section 9.3.6.4)
Messaging (Section 9.3.7.5)
R
W
MSK
A
—
PRIORITY
VECTOR
Internal (Section 9.3.7.3)
R
W
MSK
A
—
P
—
PRIORITY
VECTOR
External (Section 9.3.7.1)
R
W
MSK
A
—
P
S
—
PRIORITY
VECTOR

**Figure 9-37. Vector/Priority Register Summary**

Note the following:

- The MSK, A, PRIORITY, and VECTOR fields  have meaning only for interrupts routed to the *int* signal.
- The polarity field, P, is provided to indicate whether the signals from the corresponding source are active high or low.
- The sense field, S, is provided to allow external interrupt sources to be configured as level-sensitive so they can be routed to either *cint* or $\overline{\text{IRQ\_OUT}}$.

### 9.3.7.1 External Interrupt Vector/Priority Registers (EIVPR0–EIVPR11)

The EIVPRs, shown in Figure 9-38, contain polarity and sense fields for the external interrupts, that is, those caused by the assertion of any of IRQ[]. See Section 9.4.1, “Flow of Interrupt Control,” for information on IPR and ISR.

Offset EIVPR0: 0x0000; EIVPR1: 0x0020; EIVPR2: 0x0040; EIVPR3: 0x0060; EIVPR4: 0x0080; EIVPR5: 0x00A0; EIVPR6: 0x00C0; EIVPR7: 0x00E0; EIVPR8: 0x0100; EIVPR9: 0x0120; EIVPR10: 0x0140; EIVPR11: 0x0160

Access: Mixed

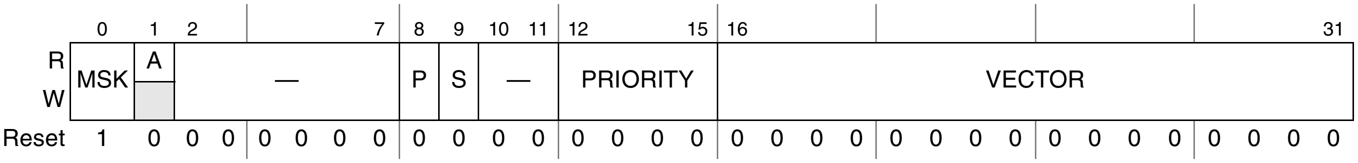


**Figure 9-38. External Interrupt Vector/Priority Registers (EIVPR0—EIVPR11)**

Table 9-38 describes the EIVPR fields.

**Table 9-38. EIVPR*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | MSK | Mask. Mask interrupts from this source. MSK affects only interrupts routed to *int*.<br>0  An interrupt request is generated if the corresponding IPR bit is set.<br>1  Further interrupts from this source are disabled. |
| 1 | A | Activity. Indicates an interrupt has been requested or is in service. The VECTOR and PRIORITY values should not be changed while this bit is set. Affects only interrupts routed to *int*.<br>0  No current interrupt activity associated with this source.<br>1  The interrupt field for this source is set in the IPR or ISR. |

Table 9-38. EIVPR*n* Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 2–7 | — | Reserved, should be cleared. |
| 8 | P | Polarity. Specifies the polarity for the external interrupt.<br>0 Polarity is active-low or negative edge-triggered.<br>1 Polarity is active-high or positive edge-triggered. |
| 9 | S | Sense. Specifies the sense for external interrupts.<br>0 The external interrupt is edge sensitive.<br>1 The external interrupt is level sensitive. This setting must be used to direct the interrupt to $\overline{\text{IRQ\_OUT}}$ or *cint*.<br>Note: If an IRQ*n* signal is used to receive INT*x* signals from one of the PCI Express ports as a root complex, S must be set to be level-sensitive. |
| 10–11 | — | Reserved, should be cleared. |
| 12–15 | PRIORITY | Priority. Specifies the interrupt priority. The lowest priority is 0 and the highest priority is 15. A priority level of 0 inhibits signalling of this interrupt to the core. Affects only interrupts routed to *int*. |
| 16–31 | VECTOR | Vector (Affects only interrupts routed to *int*). Contains the value returned when IACK is read and this interrupt resides in the corresponding interrupt request register (IRR) for that core, as shown in Figure 9-50. |

### 9.3.7.2 External Interrupt Destination Registers (EIDR0–EIDR11)

The EIDRs, shown in Figure 9-39, control the destination of external interrupts caused by the assertion of any of IRQ[]. Only one destination bit may be set; otherwise, behavior is undefined.

Offset EIDR0: 0x0010; IDR1: 0x0030; EIDR2: 0x0050; EIDR3: 0x0070; EIDR4: 0x0090; EIDR5: 0x00B0; EIDR6: 0x00D0; EIDR7: 0x00F0; EIDR8: 0x0110; EIDR9: 0x0130; EIDR10: 0x0150; EIDR11: 0x0170

Access: Read/Write

| | 0 | 1 | 2 | 3 | | | | | | | | | | | | | | | | | | | | | | | | | | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | EP | CI0 | CI1[1] | — | | | | | | | | | | | | | | | | | | | | | | | | | | | P1[1] | P0 |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

[1] Reserved in single-processor implementations.

Figure 9-39. External Interrupt Destination Registers (EIDRs)

Table 9-39 describes the EIDR fields.

Table 9-39. EIDR*n* Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0 | EP | External signal. Allows interrupt to be serviced externally. EP should be set only for level-sensitive external interrupts (EIVPR*n*[S]= 1). Setting for edge-sensitive does not provide reliable interrupt response.<br>0 Interrupt is not routed to $\overline{\text{IRQ\_OUT}}$.<br>1 Interrupt is routed to $\overline{\text{IRQ\_OUT}}$ for external service. |
| 1 | CI0 | Critical interrupt 0. CI*n* fields should be set only for level-sensitive external interrupts (EIVPR*n*[S]= 1). Setting them for edge-sensitive does not provide reliable interrupt response.<br>0 Processor core 0 does not receive this interrupt.<br>1 Directs the external interrupt to processor core 0 by causing the *cint0* output signal from the PIC to assert. See Section 9.1.2, "Interrupts to the Processor Core." |

Table 9-39. EIDR*n* Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 2 | CI1 | Critical interrupt 1. CI*n* fields should be set only for level-sensitive external interrupts (EIVPR*n*[S]= 1). Setting them for edge-sensitive does not provide reliable interrupt response. Reserved in single-processor implementations.<br>0 Processor core 1 does not receive this interrupt.<br>1 Directs the external interrupt to processor core 1 by causing the *cint1* output signal from the PIC to assert. See Section 9.1.2, “Interrupts to the Processor Core.” |
| 3–29 | — | Reserved, should be cleared. |
| 30 | P1 | Processor core 1. Indicates whether processor core 1 receives the interrupt through *int.*<br>0 Processor core 1 does not receive this interrupt.<br>1 Directs the interrupt to processor core 1 through the assertion of *int1*.<br>**Note:** Reserved in single-processor implementations. |
| 31 | P0 | Processor core 0. Indicates whether processor core 0 receives the interrupt.<br>0 Processor core 0 does not receive this interrupt.<br>1 Directs the interrupt to processor core 0 through the assertion of *int0*.<br>The default destination is for processor core 0 to receive this external interrupt after the PIC is reset. |

### 9.3.7.3 Internal Interrupt Vector/Priority Registers (IIVPR*n*)

The IIVPRs, shown in Figure 9-40, have the same fields and format as the GTVPRs, except that they apply to the internal interrupt sources listed in Table 9-3. These interrupts are all level-sensitive. See Section 9.4.1, “Flow of Interrupt Control,” for information on IPR and ISR.

**NOTE**

> Because all internal interrupts are active-high, clearing the polarity field, IIVPR*n*[P], disables that interrupt. Care should be taken to ensure this field is set during initialization and that it is not inadvertently corrupted when loading or reloading IIVPRs with priority, mask, or vector data.

Offset IIVPR0–7 0x0200, 0x0220, 0x0240, 0x0260, 0x0280, 0x02A0, 0x02C0, 0x02E0
IIVPR8–15 0x0300, 0x0320, 0x0340, 0x0360, 0x0380, 0x03A0, 0x03C0, 0x03E0
IIVPR16–23 0x0400, 0x0420, 0x0440, 0x0460, 0x0480, 0x04A0, 0x04C0, 0x04E0
IIVPR24–31 0x0500, 0x0520, 0x0540, 0x0560, 0x0580, 0x05A0, 0x05C0, 0x05E0
IIVPR32–39 0x0600, 0x0620, 0x0640, 0x0660, 0x0680, 0x06A0, 0x06C0, 0x06E0
IIVPR40–47 0x0700, 0x0720, 0x0740, 0x0760, 0x0780, 0x07A0, 0x07C0, 0x07E0

IIVPR48–55 0x0800, 0x0820, 0x0840, 0x0860, 0x0880, 0x08A0, 0x08C0, 0x08E0, IIVPR56–63
0x0900, 0x0920, 0x0940, 0x0960, 0x0980, 0x09A0, 0x09C0, 0x09E0

Access: Mixed

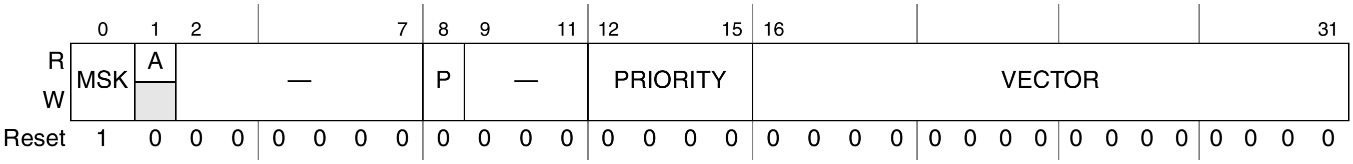


Figure 9-40. Internal Interrupt Vector/Priority Registers (IIVPRs)

Table 9-40 describes the IIVPR fields.

**Table 9-40. IIVPR*n* Field Descriptions**

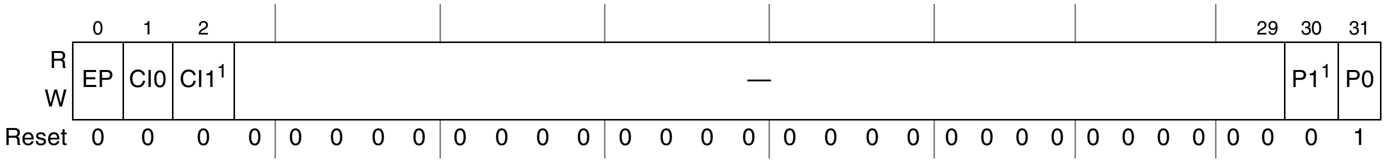
| Bits | Name | Description |
|---|---|---|
| 0 | MSK | Mask. Mask interrupts from this source. MSK affects only interrupts routed to *int*.<br>0 An interrupt request is generated if the corresponding IPR bit is set.<br>1 Further interrupts from this source are disabled. |
| 1 | A | Activity. Indicates an interrupt has been requested or is in service. The VECTOR and PRIORITY values should not be changed while this bit is set. Affects only interrupts routed to *int*.<br>0 No current interrupt activity associated with this source.<br>1 The interrupt field for this source is set in the IPR or ISR. |
| 2–7 | — | Reserved, should be cleared. |
| 8 | P | Polarity. Specifies the polarity for the internal interrupt. Note: Because all internal interrupts are active-high, clearing this bit disables the interrupt.<br>0 Interrupt polarity is active-low. This value disables the interrupt.<br>1 Interrupt polarity is active-high. |
| 9–11 | — | Reserved, should be cleared. |
| 12–15 | PRIORITY | Priority. Specifies the interrupt priority. The lowest priority is 0 and the highest priority is 15. A priority level of 0 inhibits signalling of this interrupt to the core. Affects only interrupts routed to *int*. |
| 16–31 | VECTOR | Vector (Affects only interrupts routed to *int*). Contains the value returned when IACK is read and this interrupt resides in the corresponding interrupt request register (IRR) for that core, as shown in Figure 9-50. |

### 9.3.7.4 Internal Interrupt Destination Registers (IIDR*n*)

The IIDRs, shown in Figure 9-41, have the same fields and format as EIVDRs, except that they apply to the internal interrupt sources listed in Table 9-3. Only one destination bit may be set; otherwise, behavior is undefined.

Offset IIDR0–70x0210, 0x0230, 0x0250, 0x0270, 0x0290, 0x02B0, 0x02D0, 0x02F0
IIDR8–15 0x0310, 0x0330, 0x0350, 0x0370, 0x0390, 0x03B0, 0x03D0, 0x03F0
IIDR16–23 0x0410, 0x0430, 0x0450, 0x0470, 0x0490, 0x04B0, 0x04D0, 0x04F0
IIDR24–31 0x0510, 0x0530, 0x0550, 0x0570, 0x0590, 0x05B0, 0x05D0, 0x05F0
IIDR32–39 0x0610, 0x0630, 0x0650, 0x0670, 0x0690, 0x06B0, 0x06D0, 0x06F0
IIDR40–47 0x0710, 0x0730, 0x0750, 0x0770, 0x0590, 0x07B0, 0x07D0, 0x07F0
IIDR48–53 0x0810, 0x0830, 0x0850, 0x0870, 0x0890, 0x08B0, 0x08D0, 0x08F0
IIDR54–63 0x0910, 0x0930, 0x0950, 0x0970, 0x0990, 0x09B0, 0x09D0, 0x09F0

Access: Read/Write


0
1
2
29
30
31
R
W
EP
CI0
CI1[1]
—
P1[1]
P0
Reset 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

[1] Reserved in single-processor implementations.

**Figure 9-41. Internal Interrupt Destination Registers (IIDRs)**

Table 9-41 describes the IIDR fields.

**Table 9-41. IIDR*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | EP | External signal. Allows internal interrupt to be serviced externally.<br>0 Interrupt is not routed to $\overline{\text{IRQ\_OUT}}$.<br>1 Interrupt is routed to $\overline{\text{IRQ\_OUT}}$ for external service. |
| 1 | CI0 | Critical interrupt 0. See Section 9.1.2, "Interrupts to the Processor Core," for more information.<br>0 Processor core 0 does not receive this interrupt.<br>1 Directs the internal interrupt to processor core 0 by causing the *cint0* output signal from the PIC to assert. |
| 2 | CI1 | Critical interrupt 1. See Section 9.1.2, "Interrupts to the Processor Core," for more information. Reserved in single-processor implementations.<br>0 Processor core 1 does not receive this interrupt.<br>1 Directs the internal interrupt to processor core 1 by causing the *cint1* output signal from the PIC to assert. |
| 3–29 | — | Reserved, should be cleared. |
| 30 | P1 | Processor core 1. Indicates whether processor core 1 receives the interrupt through *int.*<br>0 Processor core 1 does not receive this interrupt.<br>1 Directs the interrupt to processor core 1 through the assertion of *int1*.<br>**Note:** Reserved in single-processor implementations. |
| 31 | P0 | Processor core 0. Indicates whether processor core 0 receives the interrupt.<br>0 Processor core 0 does not receive this interrupt.<br>1 Directs the interrupt to processor core 0 through the assertion of *int0*.<br>The default destination is for processor core 0 to receive this external interrupt after the PIC is reset. |

### 9.3.7.5 Messaging Interrupt Vector/Priority Registers (MIVPR*n*)

The MIVPRs have the same fields and format as the GTVPRs, except they apply to messaging interrupts. See Section 9.4.1, "Flow of Interrupt Control," for information on IPR and ISR.

Offset MIVPR0: 0x1600; MIVPR1: 0x1620; MIVPR2: 0x1640; MIVPR3: 0x1660
MIVPR4: 0x1680; MIVPR5: 0x16A0; MIVPR6: 0x16C0; MIVPR7: 0x16E0

Access: Mixed

| | 0 | 1 | 2–11 | 12–15 | 16–31 |
|---|---|---|---|---|---|
| R | MSK | A | — | PRIORITY | VECTOR |
| W | MSK | | — | PRIORITY | VECTOR |
| Reset | 1 | 0 | 0 0 0 0 0 0 0 0 0 0 | 0 0 0 0 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 |

**Figure 9-42. Messaging Interrupt Vector/Priority Registers (MIVPR*n*)**

Table 9-42 describes the MIVPR*n* fields.

**Table 9-42. MIVPR*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | MSK | Mask. Mask interrupts from this source. MSK affects only interrupts routed to *int*.<br>0 An interrupt request is generated if the corresponding IPR bit is set.<br>1 Further interrupts from this source are disabled. |
| 1 | A | Activity. Indicates an interrupt has been requested or is in service. The VECTOR and PRIORITY values should not be changed while this bit is set. Affects only interrupts routed to *int*.<br>0 No current interrupt activity associated with this source.<br>1 The interrupt field for this source is set in the IPR or ISR. |

Table 9-42. MIVPR*n* Field Descriptions (continued)

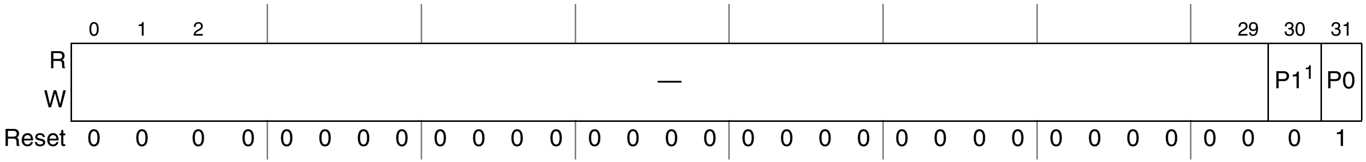
| Bits | Name | Description |
|---|---|---|
| 2–11 | — | Reserved, should be cleared. |
| 12–15 | PRIORITY | Priority. Specifies the interrupt priority. The lowest priority is 0 and the highest priority is 15. A priority level of 0 inhibits signalling of this interrupt to the core. Affects only interrupts routed to *int*. |
| 16–31 | VECTOR | Vector (Affects only interrupts routed to *int*). Contains value returned when IACK is read and this interrupt resides in the corresponding interrupt request register (IRR) for that core, as shown in Figure 9-50. |

### 9.3.7.6 Messaging Interrupt Destination Registers (MIDR0–MIDR7)

The messaging interrupt destination registers (MIDRs), shown in Figure 9-43, control the destination for the messaging interrupts. Only one destination bit may be set; otherwise, behavior is undefined.

Offset MIDR0: 0x1610; MIDR1: 0x1630; MIDR2: 0x1650; MIDR3: 0x1670
MIDR0: 0x1690; MIDR1: 0x16B0; MIDR2: 0x16D0; MIDR3: 0x16F0

Access: Read/Write



| | 0–29 | 30 | 31 |
|---|---|---|---|
| R<br>W | — | P1[1] | P0 |
| Reset | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 | 1 |

[1] Reserved in single-processor implementations.

Figure 9-43. Messaging Interrupt Destination Registers (MIDR*n*)

Table 9-43 describes the MIDR*n* fields.

Table 9-43. MIDR*n* Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–29 | — | Reserved, should be cleared. |
| 30 | P1 | Processor core 1. Indicates whether processor core 1 receives the interrupt through *int*.<br>0 Processor core 1 does not receive this interrupt.<br>1 Directs the interrupt to processor core 1 through the assertion of *int1*.<br>**Note:** Reserved in single-processor implementations. |
| 31 | P0 | Processor core 0. Indicates whether processor core 0 receives the interrupt.<br>0 Processor core 0 does not receive this interrupt.<br>1 Directs the interrupt to processor core 0 through the assertion of *int0*.<br>The default destination is for processor core 0 to receive this external interrupt after the PIC is reset. |

### 9.3.8 Per-CPU (Private Access) Registers

The OpenPIC programming model supports multiprocessor systems of up to 32 separate processors. As such, the OpenPIC interface specification provides for coordinating both the requesting and servicing of interrupts among several processor cores within a single integrated device. To comply with the OpenPIC specification, the PIC incorporates several of these multiprocessor capabilities.

**NOTE**

Note that these registers are meaningful only for interrupts routed to *int*.

The registers in Table 9-44 are called per-CPU registers because they are duplicated for each core in a multi-core device. The OpenPIC interface specifies that a copy of these registers be available to each core at the same physical address by using the ID of the processor core that initiates the transaction to determine the set of per-CPU registers to access.

**Table 9-44. Per-CPU Registers—Private Access Address Offsets**

| Register Name | Offset |
|---|---|
| Interprocessor 0 dispatch register (IPIDR0) | 0x0040 |
| Interprocessor 1 dispatch register (IPIDR1) | 0x0050 |
| Interprocessor 2 dispatch register (IPIDR2) | 0x0060 |
| Interprocessor 3 dispatch register (IPIDR3) | 0x0070 |
| Current task priority register (CTPR) | 0x0080 |
| Who am I register (WHOAMI0) | 0x0090 |
| Interrupt acknowledge register (IACK) | 0x00A0 |
| End of interrupt register (EOI) | 0x00B0 |

These addresses, shown in Table 9-44, appear in the memory map at the same offset for every processor in what is called the private access space. This duplication allows user code to execute correctly in an multiprocessor environment without needing to know which core it is running on. On a single-core device, each register has two addresses, one in the normal address space and one in the private access space. It is included on even single-core devices to simplify the porting of such code.

Figure 9-44 shows how the duplicated registers are addressed in a four-core device Note that when accessing a register normally, each core sources a different address. However, when accessing the same register using the per-CPU address space, each core sources the same address.

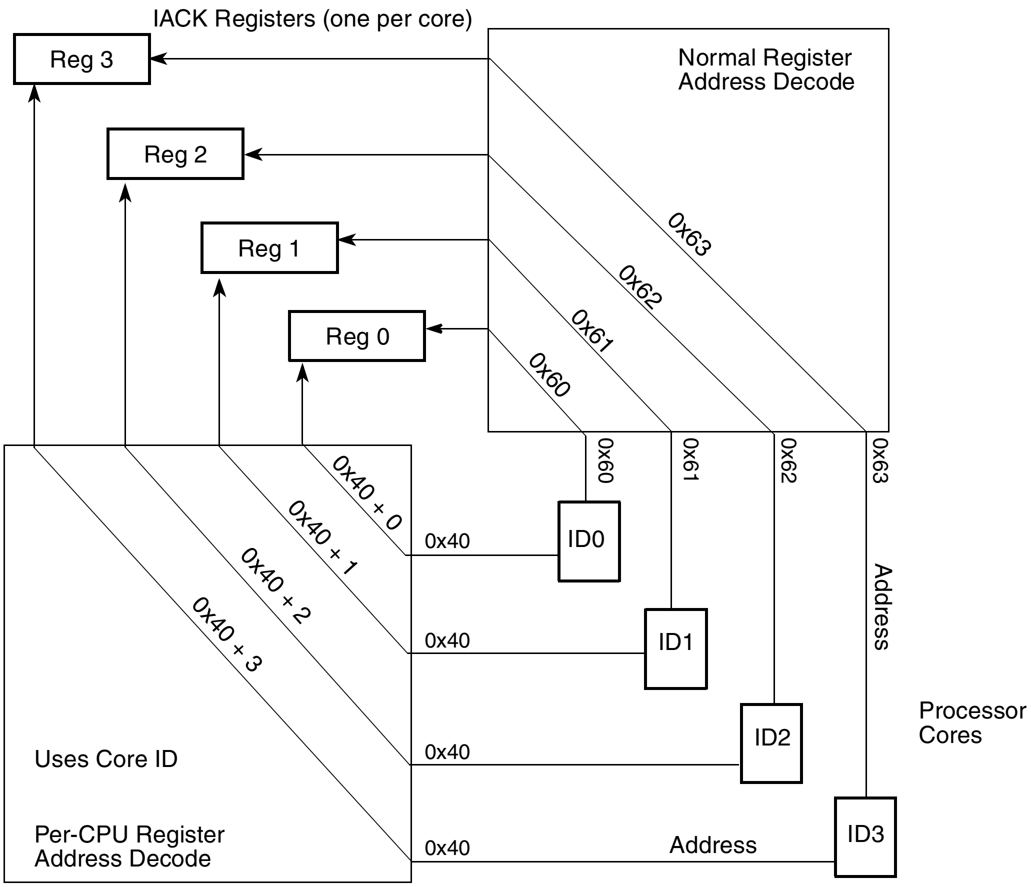


**Figure 9-44. Per-CPU Register Address Decoding in a Four-Core Device**

### 9.3.8.1 Interprocessor Interrupt Dispatch Register (IPIDR0–IPIDR3)

Figure 9-45 shows the four IPIDRs, one for each interprocessor interrupt channel. Writing to an IPIDR with a bit set causes a self interrupt for a single-core device. Because external bus masters can write to these registers, this feature can serve as a doorbell type interrupt.

Offset Processor core 0: IPIDR0: 0x0040; IPIDR1: 0x0050; IPIDR2: 0x0060; IPIDR3: 0x0070
Processor core 1[1]:IPIDR0: 0x1040; IPIDR1: 0x1050; IPIDR2: 0x1060; IPIDR3: 0x1070
Pre-CPU offsets: IPIDR0: 0x0040; IPIDR1: 0x0050; IPIDR2: 0x0060; IPIDR3: 0x0070

Access: Write Only

| | 0 | 30 | 31 |
|---|---|---|---|
| R | — | | |
| W | | P1[1] | P0 |
| Reset | All zeros | | |

[1] Reserved in single-processor implementations.

**Figure 9-45. Interprocessor Interrupt Dispatch Registers (IPIDR0–IPIDR3)**

Table 9-40 describes the IPIDR*n* fields.

**Table 9-45. IPIDR*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–29 | — | Reserved, should be cleared. |
| 30 | P1 | Processor core 1. Specifies if processor core 1 receives the interrupt. This interrupt is multicasting, so both P0 and P1 can be set.<br>0 Processor core 1 does not receive the interrupt<br>1 Directs the interrupt to processor core 1<br>**Note:** Reserved in single-processor implementations. |
| 31 | P0 | Processor core 0. Determines if processor core 0 receives the interrupt.<br>0 Processor core 0 does not receive the interrupt.<br>1 Directs the interrupt to processor core 0. |

## 9.3.8.2 Processor Core Current Task Priority Registers 0–1 (CTPR0–CTPR1)

There is one CTPR per processor core on this device as shown in Figure 9-46.

Offset CTPR0: 0x0080; CTPR1[1] 0x1080; Pre-CPU offset: 0x0080 Access: Read/Write

| | 0–27 | 28–31 |
|---|---|---|
| R<br>W | — | TASKP |
| Reset | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 1 1 1 1 |

[1] Reserved in single-processor implementations.

**Figure 9-46. Processor Core Current Task Priority Registers (CTPR*n*)**

### NOTE

CTPR has meaning only for interrupts routed to *int*.

Software must write the priority of the current processor core task in the CTPR for each core. The PIC uses this value for comparison with the priority of incoming interrupts. Given several concurrent incoming interrupts, the highest priority interrupt is asserted to that core if the following apply:

- The interrupt is not masked.
- The priority of the interrupt is higher than the values in the corresponding CTPR[TASKP] and ISR.

Table 9-46 describes the CTPR task priority field.

**Table 9-46. CTPR*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–27 | — | Reserved, should be cleared. |
| 28–31 | TASKP | Task priority. Indicates the threshold that individual interrupt priorities must exceed for the interrupt request to be serviced.<br>0000–1111 *x*VPR*n*[PRIORITY] must exceed this value for the interrupt request to be serviced. Note the following special cases:<br>0000 Lowest priority. All interrupts except those whose priority are 0 can be serviced.<br>1111 Highest priority. No interrupts are signaled to that processor core. Hardware selects this value on a device hard reset or when the corresponding PIR[P*n*] is set. |

### 9.3.8.3 Who Am I Registers 0–1 (WHOAMI0–WHOAMI1)

The processor core WHOAMI*n* register, shown in Figure 9-47, can be read by a processor core to determine its physical connection to the PIC. The value returned when reading this register may be used to determine the value for the destination masks used for dispatching interrupts.

Offset WHOAMI: 0x0090; WHOAMI1[1]: 0x1090; Per-CPU offset: 0x0090 Access: Read only

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | — | | | | | | | | | | | | | | | | | | | | | | | | | | | ID | | | | |
| W | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | *n* | *n* | *n* | *n* | *n* |

[1] Reserved in single-processor implementations.

**Figure 9-47. Processor Core Who Am I Registers (WHOAMI*n*)**

Table 9-47 describes the WHOAMI*n* fields.

**Table 9-47. WHOAMI*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–26 | — | Reserved, should be cleared. |
| 27–31 | ID | Returns the ID of the processor core reading this register.<br>0_0000 Processor core 0<br>0_0001 Processor core 1. (Value not supported in single-processor implementations.)<br>1_1111 Other devices |

### 9.3.8.4 Processor Core Interrupt Acknowledge Registers 0–1 (IACK0–IACK1)

**NOTE**

> IACK has meaning only for interrupts routed to *int* and should not be accessed for interrupts routed to *cint* or $\overline{\text{IRQ\_OUT}}$.

In systems based on processors built on Power Architecture™ technology, the interrupt acknowledge function occurs as an explicit read operation to a memory-mapped interrupt acknowledge register (IACK), shown in Figure 9-48. Each processor core has an IACK register assigned to it. Reading IACK returns the interrupt vector corresponding to the highest priority pending interrupt. Reading IACK also has the following side effects:

- The associated field in the corresponding interrupt pending register (IPR) is cleared for edge-sensitive interrupts. See Section 9.4.1.2, “Interrupts Routed to int.”
- The corresponding in-service register (ISR) is updated.
- The corresponding *int* output signal from the PIC is negated.

Reading IACK when no interrupt is pending returns the spurious vector value, as described in Section 9.3.1.8, “Spurious Vector Register (SVR).”

Offset IACK0: 0x00A0; IACK1[1]: 0x10A0; Per-CPU offset: 0x00A0 Access: Read only

|  | 0–15 | 16–31 |
|---|---|---|
| R | — | VECTOR |
| W |  | ▨ |
| Reset | All zeros |  |

[1] Reserved in single-processor implementations.

**Figure 9-48. Processor Core Interrupt Acknowledge Registers (IACK*n*)**

Table 9-48 describes the IACK*n* fields.

**Table 9-48. IACK*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–15 | — | Reserved, should be cleared. |
| 16–31 | VECTOR | Interrupt vector. Vector of the highest pending interrupt (read only) |

### 9.3.8.5 Processor Core End of Interrupt Registers (EOI0–EOI1)

**NOTE**

EOI has meaning only for interrupts routed to *int* and should not be accessed for interrupts routed to *cint* or $\overline{\text{IRQ\_OUT}}$.

Each core is assigned an EOI register, shown in Figure 9-49. Writing to EOI signals the end of processing for the highest-priority interrupt (routed to *int*) currently in service. It also updates the corresponding ISR*n* by retiring the highest priority interrupt. Data values written to EOI are ignored, and zero is assumed.

Offset EOI0: 0x00B0; EOI1[1]: 0x10B0; Per-CPU offset: 0x00B0 Access: Write Only

|  | 0–27 | 28–31 |
|---|---|---|
| R | — | ▨ |
| W |  | EOI CODE |
| Reset | All zeros |  |

[1] Reserved in single-processor implementations.

**Figure 9-49. End of Interrupt Registers (EOI*n*)**

Table 9-49 describes the EOI*n* fields.

**Table 9-49. EOI*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–27 | — | Reserved, should be cleared. |
| 28–31 | EOI CODE | 0000 (write only) |

## 9.4 Functional Description

This section is a functional description of the PIC.

### 9.4.1 Flow of Interrupt Control

Figure 9-50 shows the flow of interrupts directed by the PIC to the *int*, *cint*, and $\overline{\text{IRQ\_OUT}}$ outputs. Note that this diagram describes a conceptual model of an PIC on a single processor. This logic is replicated for each implemented processor. This conceptual diagram does not fully represent all internal circuitry of the implementation.

This figure focusses especially on the OpenPIC-defined logic and shows how the PIC controls interrupt requests that target the *int* signal. The flow in Figure 9-50 is from the bottom to the top, and shows at the bottom how the destination register associated with each source determines the path.

#### 9.4.1.1 Interrupts Routed to *cint* or $\overline{\text{IRQ\_OUT}}$

Interrupt requests routed to *cint* or $\overline{\text{IRQ\_OUT}}$ bypass the logic that is dedicated to interrupt sources that compete for *int*. That is, if *x*IDR*n*[CI*n*] or *x*IDR*n*[EP] = 1, corresponding *x*IVPR field settings have no hardware effects; however, an interrupt handler may be able to make use of some of those fields.

*cint* signals are connected to the respective core's critical interrupt input.

**NOTES**

Because interrupt sources routed to *cint* or $\overline{\text{IRQ\_OUT}}$ must be level sensitive, EIVPR[S] should be set. See Section 9.3.7.1, "External Interrupt Vector/Priority Registers (EIVPR0–EIVPR11)."

Because these interrupts bypass the OpenPIC logic, it is especially important that handlers do not read IACK. Doing so causes a spurious interrupt. Likewise, they should not write EOI.

#### 9.4.1.2 Interrupts Routed to *int*

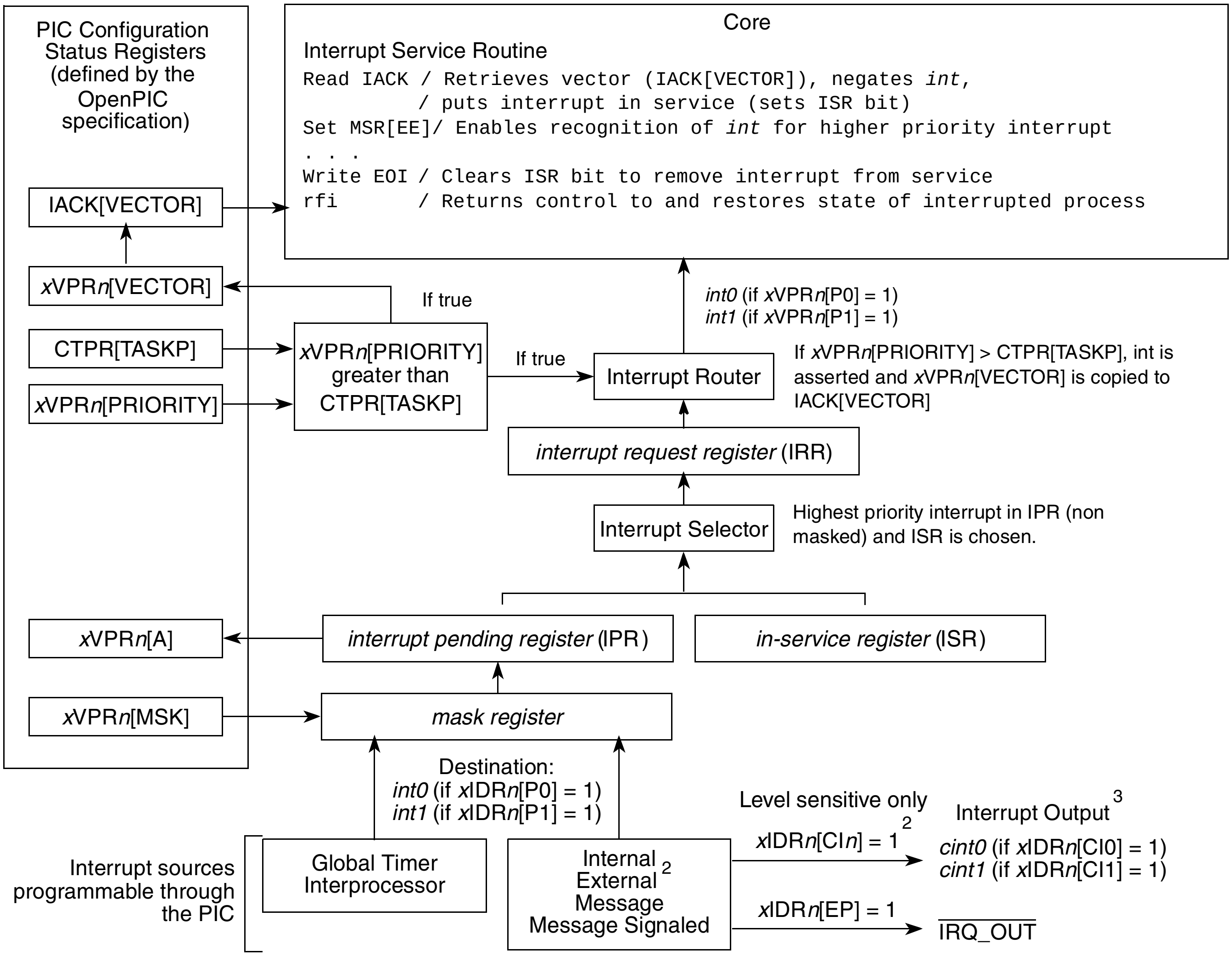
As shown in Figure 9-50, the PIC receives interrupt requests from external and internal sources and from within the PIC itself. As Figure 9-50 shows, all of these interrupt sources can be routed to *int*; the global timer and timer processor interrupts can be directed only to *int*.

The sources' mask bits (*x*VPR*n*[MSK]) are tracked in the internal mask register. If a source's MSK bit is set, the mask register prevents the PIC from asserting *int* on its behalf.

Unmasked interrupt requests are qualified and latched in the interrupt pending register (IPR), an internal interrupt summary register with a bit for each source. If an interrupt request is multi-cast, a bit is set in the IPR for each targeted processor. Although the IPR cannot be read by software, when an IPR bit is set, the corresponding source's activity bit (*x*VPR*n*[A]) is automatically set.

The interrupt selector monitors the IPR and the in-service register (ISR), which tracks previously taken interrupts that were superseded by a higher-priority interrupt before the interrupt handler finished. The interrupt selector recognizes the highest priority unmasked interrupt request and latches it into the interrupt request register (IRR). The source's vector (*x*VPR*n*[VECTOR]) is copied to IACK[VECTOR], which the interrupt handler retrieves by reading IACK.

If the priority (*x*VPR*n*[PRIORITY]) of an interrupt latched in the IRR is higher than the value in the target processor's CTPR[TASKP], the interrupt router asserts the external interrupt signal (*int*), causing that processor core to vector to its external interrupt handler.


PIC Configuration Status Registers (defined by the OpenPIC specification)
IACK[VECTOR]
xVPRn[VECTOR]
CTPR[TASKP]
xVPRn[PRIORITY]
xVPRn[A]
xVPRn[MSK]
Core
Interrupt Service Routine
Read IACK / Retrieves vector (IACK[VECTOR]), negates int,
/ puts interrupt in service (sets ISR bit)
Set MSR[EE]/ Enables recognition of int for higher priority interrupt
. . .
Write EOI / Clears ISR bit to remove interrupt from service
rfi / Returns control to and restores state of interrupted process
If true
xVPRn[PRIORITY] greater than CTPR[TASKP]
If true
Interrupt Router
int0 (if xVPRn[P0] = 1)
int1 (if xVPRn[P1] = 1)
If xVPRn[PRIORITY] > CTPR[TASKP], int is asserted and xVPRn[VECTOR] is copied to IACK[VECTOR]
interrupt request register (IRR)
Interrupt Selector
Highest priority interrupt in IPR (non masked) and ISR is chosen.
interrupt pending register (IPR)
in-service register (ISR)
mask register
Destination:
int0 (if xIDRn[P0] = 1)
int1 (if xIDRn[P1] = 1)
Level sensitive only
xIDRn[CIn] = 1 2
xIDRn[EP] = 1
Interrupt Output 3
cint0 (if xIDRn[CI0] = 1)
cint1 (if xIDRn[CI1] = 1)
IRQ_OUT
Interrupt sources programmable through the PIC
Global Timer Interprocessor
Internal External Message Message Signaled 2

[1] If *cint* or IRQ_OUT is the destination, EIVPR*n*[S] must be set to configure the source as level sensitive.

[2] If multiple destination register bits are set, PIC behavior is undefined.

[3] Although setting CI*n* directs the interrupt request to the critical interrupt output (*cint0*/*cint1*), integrated logic may connect this signal to a different interrupt input to the core.

**Figure 9-50. PIC Interrupt Processing Flow Diagram for Each Core (*n*)**

The interrupt handler must acknowledge the interrupt by explicitly reading the corresponding IACK register, described in Section 9.3.8.4, "Processor Core Interrupt Acknowledge Registers 0–1 (IACK0–IACK1)." The PIC interprets this read as an interrupt acknowledge (IACK) cycle. The IACK cycle not only returns the source's vector, it also negates the *int* signal to the processor (making it possible for a higher priority interrupt to assert *int*) and sets the source's bit in the ISR, indicating that this interrupt has been put in service. An interrupt remains in service from the time until the corresponding end-of-interrupt register (EOI) is written, generating what the PIC considers an EOI signal.

Figure 9-50 shows required elements in the interrupt handler

#### 9.4.1.2.1 Interrupt Source Priority

Each interrupt source routed to *int* is assigned a value through its *x*VPR*n*[PRIORITY] field. Priority values range from 0 to 15, where 15 is the highest. Interrupts are delivered only when the priority of the source is greater than the destination processor's CTPR[TASKP]. Therefore, setting *x*VPR*n*[PRIORITY] to zero inhibits that interrupt. Likewise, setting TASKP to 15 prevents the PIC from delivering interrupts to that core through the *int* signal. Note that this is the reset value, preventing the PIC from asserting *int* before the PIC is configured.

The PIC services simultaneous interrupts occurring with the same priority according to the following order:

1. MSG0–MSG7
2. MSI0–MSI7
3. IPI0–IPI3
4. Group A timer0–timer3
5. Group B timer0–timer3
6. IRQ[]/PCI INT*x*
7. Internal0–internal63

For example, if MSG0, MSG2, and IPI0 are all assigned the same priority and receive simultaneous interrupts, they are serviced in the following order:

1. MSG0
2. MSG2
3. IPI0

#### 9.4.1.2.2 Interrupt Acknowledge

When the PIC causes *int* to be asserted, the external interrupt service routine acknowledges the request by reading that core's IACK register, which at this point holds the 16-bit vector value for the interrupt source that generated the request. This is the value programmed in that source's *x*VPR*n*[VECTOR]. Reading IACK has the following effects:

- The *int* signal for that core is negated, making it possible for another interrupt source to signal an external interrupt to the core, and more particularly, allowing the PIC to signal a higher-priority interrupt, as described in Section 9.4.1.2.4, "Nesting of Interrupts."
- The source that caused that resource is represented in the internal in-service register (ISR).

The interrupt is then considered to be in service. It remains so until the processor core performs a write to the corresponding EOI. Writing to EOI is referred to as an EOI cycle.

#### 9.4.1.2.3 Spurious Vector Generation

Under certain circumstances, the PIC has no valid vector to return to a processor core during an interrupt acknowledge cycle. In these cases, the spurious vector from the spurious vector register is returned. The following cases cause a spurious vector fetch:

- *int* is asserted in response to an externally or internally-sourced interrupt which is activated with level-sensitive logic, and the asserted level is negated before the interrupt is acknowledged.
- *int* is asserted for an interrupt source that is later masked (using the mask bit in the vector/priority register corresponding to that source) before the interrupt is acknowledged.
- *int* is asserted for an interrupt source that is later masked by an increase in the task priority level before the interrupt is acknowledged.
- An interrupt acknowledge cycle is performed by the processor core in spite of the fact that the int signal has not been asserted by the PIC.

In all cases, a spurious vector is returned only if no pending interrupt has sufficient priority to signal an interrupt, otherwise, the vector for that interrupt source is returned.

**NOTE**

EOI should not be written in response to a spurious vector. Otherwise, a previously accepted interrupt might be cleared unintentionally.

#### 9.4.1.2.4 Nesting of Interrupts

While an interrupt is being handled, if an interrupt request arrives with a higher *x*VPR*n*[PRIORITY] value, the interrupt being serviced is superseded. As described in Section 9.4.1.2, “Interrupts Routed to *int*,” the PIC asserts *int*, and the newer, higher priority interrupt is handled. This happens even if software, as part of its interrupt service routine, updates the corresponding CTPR with a lower value.

Thus, although several interrupts can be in service simultaneously (and tracked by the ISR), the highest priority interrupt by that processor is always the one actively handled. When the interrupt routine completes, it performs a write EOI cycle, a side effect of which is to take the current highest priority interrupt out of service (removes it from the ISR). At this point, the interrupt selector chooses the new highest priority interrupt request, and, assuming CTPR[TASKP] has not been updated to a value higher than the new interrupt, the PIC asserts *int* on its behalf.

The next write EOI cycle takes the current highest priority interrupt out of service. An interrupt with lower priority than those in service is not started until all higher priority interrupts complete even if its priority is greater than the CTPR value.

### 9.4.2 Interprocessor Interrupts

Processors 0 and 1 can generate interprocessor interrupts that target either or both processors. A self interrupt occurs when a core dispatches an interprocessor interrupt event to itself. Interrupts are initiated by writing either or both of the PO*n* bits in an interprocessor interrupt dispatch register (IPIDR0–IPIDR3) of one of the four IPI channels. If subsequent interprocessor interrupts from a given channel to a given target processor are initiated before the first is acknowledged, only one interrupt is generated.

### 9.4.3 Message Interrupts

The eight MSGRs, described in Section 9.3.5.1, “Message Registers (MSGR0–MSGR7),” can be used to send 32-bit messages to one or more processors. A messaging interrupt is generated by writing an MSGR if the corresponding MER bit is set and the interrupt is not masked. Reading a MSGR or writing a 1 to the status bit clears the interrupt.

### 9.4.4 Shared Message Signaled Interrupts

There are eight shared MSIRs, described in Section 9.3.6.1, “Shared Message Signaled Interrupt Registers (MSIR0–MSIR7),” that indicate which of the interrupt sources sharing the MSI register have pending interrupts. Up to 32 sources can share any individual MSI register. A shared message signaled interrupt is generated by writing to Shared Message Signaled Interrupt Index Register (MSIIR) fields SRS and IBS. This register is primarily intended to support inbound PCI Express message signaled interrupts (MSIs) when the PCI Express controller is configured as a root complex (RC).

MSIIR[SRS] selects the associated MSIR and MSIIR[IBS] selects the interrupt flag/bit in that register that is to be set. The corresponding interrupt needs to be unmasked for the interrupt to occur. A read to an MSIR clears the all of its flags.

### 9.4.5 PCI Express INTx/IRQ*n* Sharing

Whenever the PCI Express controller is in root complex mode and it receives an inbound INTx asserted or negated message transaction, it asserts or negates an equivalent internal INTx signal to the PIC. This INTx virtual-wire interrupt signaling mechanism replaces the PCI standard sideband interrupts (INTA, INTB, INTC, and INTD) that historically were connected to the IRQ*n* external interrupt inputs. The internal INTx signals from the PCI Express controller are logically combined with the interrupt request (IRQ*n*) signals so that they share the same OpenPIC external interrupt controlled by the associated EIVPR*n* and EIDR*n* registers.

Table 9-50 details the association of INTx signals to IRQ*n* signals.

**Table 9-50. PCI Express INTx/IRQ*n* Sharing**

| PCI Express Number | INTx | IRQ*n* |
|---|---|---|
| PCI Express 1 | INTA | IRQ0 |
| | INTB | IRQ1 |
| | INTC | IRQ2 |
| | INTD | IRQ3 |
| PCI Express 2 | INTA | IRQ4 |
| | INTB | IRQ5 |
| | INTC | IRQ6 |
| | INTD | IRQ7 |

Table 9-50. PCI Express INTx/IRQ*n* Sharing (continued)

| PCI Express Number | INTx | IRQ*n* |
|---|---|---|
| PCI Express 3 | INTA | IRQ8 |
| | INTB | IRQ9 |
| | INTC | IRQ10 |
| | INTD | IRQ11 |

In general, these signals should be considered mutually exclusive. If a PCI Express INTx signal is being used, the PIC must be configured so that external interrupts are level sensitive (EIVPRn[S] = 1). If an IRQ*n* signal is being used as edge-triggered (EIVPRn[S] = 0), the system must not allow inbound PCI Express INTx transactions.

Note that it is possible to share IRQ*n* and INTx if the external interrupt is level sensitive; however, if an interrupt occurs, the interrupt service routine must poll both the external sources connected to the IRQ*n* input and the PCI Express INTx sources to determine from which path the external interrupt came. In any case, IRQ*n* should be pulled to the negated state as determined by the associated polarity setting in EIVPR*n*[P].

## 9.4.6 Global Timers

There are appropriate clock prescalers and synchronizers to provide a time base for the internal PIC timers. These 8 timers are organized as 2 groups of 4 timers each. The timers can be individually programmed to generate a processor core interrupt when they count down to zero and can be used to generate regular periodic interrupts. Each timer has the following four configuration and control registers:

- Global timer current count register (GTCCR*xn*)
- Global timer base count register (GTBCR*xn*)
- Global timer vector-priority register (GTVPR*xn*)
- Global timer destination register (GTDR*xn*)

The timer frequency should be written to the TFRR*xn*, described in Section 9.3.2.1, “Timer Frequency Reporting Register (TFRRA–TFRRB).”

Timer interrupts are all edge-triggered interrupts. If a timer period expires while a previous interrupt from the same source is pending or in service, the subsequent interrupt is lost.

The timer control register (TCR) provides users with the ability to create timers larger than the 31-bit global timers. The timer frequency can also be changed by setting the appropriate TCR fields, as described in Section 9.3.2.6, “Timer Control Registers (TCRA–TCRB).”

## 9.4.7 Resets

This section describes the behavior of the PIC at reset and the PIC’s ability to initiate processor resets.

### 9.4.8 Resetting the PIC

The PIC is reset by a device power-on reset (POR) or by software that sets GCR[RST], either of which causes the following:

- All pending and in-service interrupts are cleared.
- All interrupt mask bits are set.
- Polarity, sense, external signal, critical interrupt, and activity fields are reset to default values.
- PIR, TFRR, TCR, MER, MSR, and MSGR0–MSGR7 are cleared.
- MSG and timer destination fields are set.
- The interprocessor dispatch registers are cleared.
- All timer base count values are reset to zero with count inhibited.
- CTPR[TASKP] is reset to 0x000F, disabling delivery of interrupts that target *int*.
- The spurious interrupt vector resets to 0xFFFF.
- The PMMRs are reset to 0xFFFF.
- The PIC defaults to the pass-through mode (GCR[M] = 0).
- All other registers remain at their pre-reset programmed values.

GCR[RST] is automatically cleared when the reset sequence is complete.

#### 9.4.8.1 Processor Core Initialization

A software reset can be routed to either of the cores by writing to the processor core initialization register (PIR). This causes the assertion of the corresponding *core_hreset* output signal from the PIC. When this occurs, the corresponding CTPR also gets written to 0x000F to prevent delivery of any interrupts to *int*.

## 9.5 Initialization/Application Information

This section contains initialization and application information for the PIC.

### 9.5.1 Programming Guidelines

The following subsections contain information about programming PIC registers.

#### 9.5.1.1 PIC Registers

Most PIC control and status registers are readable and return the last value written. The exceptions to this rule are as follows:

- Interprocessor dispatch and EOI registers, which return zeros on reads.
- Activity bits (A) of the vector/priority registers reflect the status of the corresponding interrupt source.
- IACK, which returns the vector of the highest priority pending interrupt or the spurious vector (SVR[VECTOR]) if none is pending.
- Reserved fields always return 0.

When the PIC is in mixed mode (GCR[M] = 1), the following guidelines are recommended:

- All PIC registers must be located in a cache-inhibited, guarded area (configured through the core's MMU).
- The PIC portion of the address map must be set up appropriately.

In addition, the following initialization sequence is recommended:

1. Write the vector, priority, and polarity values in each interrupt's vector/priority register, leaving their MSK (mask) bit set. This is required only if interrupts are used.
2. Clear CTPR (CTPR = 0x0000_0000).
3. Program the PIC to mixed mode by setting GCR[M].
4. Clear the MSK bit in the vector/priority registers to be used.
5. Perform a software loop to clear all pending interrupts:
   - Load counter with FRR[NIRQ].
   - While counter > 0, read IACK and write EOI to guarantee all the IPR and ISR bits are cleared.
6. Set the processor core CTPR values to the desired values.
7. Read the MSIRs to clear any pending message signaled interrupts that may have been pending before a soft reset.
8. Set MER to 0x0000_000F. See Section 9.3.5.2, "Message Enable Register (MER)," for more information.

Depending on the interrupt system configuration, the PIC may generate spurious interrupts to clear interrupts latched during power-up. A spurious or non-spurious vector is returned for an interrupt acknowledge cycle in this case. See the programming note below for the non-spurious case.

**NOTE**

Because the default polarity/sense for external interrupts is edge-sensitive, and edge-sensitive interrupts are not cleared until they are acknowledged, it is possible for the PIC to store spurious edges detected during power-up as pending external interrupts. If software permanently configures an external interrupt source to be edge-sensitive, it may receive the vector for the interrupt source and not a spurious interrupt vector when software clears the mask bit. This can occur once for any edge-sensitive interrupt when its mask bit is first cleared and the PIC is in mixed mode.

To avoid a false interrupt for this case, software can clear the IPR of these spurious edge detections by first configuring the polarity/sense of external interrupt sources to be level-sensitive: high-level if the input is a positive-edge source and low-level if it is a negative-edge source (while the mask bit remains set). After this is complete, configuring the external interrupt source as edge-sensitive does not cause a false interrupt.

### 9.5.1.2 Changing Interrupt Source Configuration

To change the vector, priority, polarity, sense, or destination of an active (unmasked) interrupt source, the following steps should be taken:

1. Mask the source using the mask (MSK) bit in the vector/priority register.
2. Wait for the activity (A) bit for that source to be cleared.
3. Make the desired changes.
4. Unmask the source.

Note that changing the destination from *int* to *cint* or $\overline{\text{IRQ\_OUT}}$ makes the A, MSK, and PRIORITY fields meaningless.

# Chapter 10
# Security Engine (SEC) 3.0

This chapter describes the functionality of Freescale’s integrated security engine (SEC 3.0). It addresses the following topics:

- Section 10.1, “SEC Architecture Overview”
- Section 10.2, “Configuration of Internal Memory Space”
- Section 10.3, “Descriptors”
- Section 10.4, “Polychannel”
- Section 10.5, “Controller”
- Section 10.5.1, “Bus Transfers”
- Section 10.6, “Power Saving Mode
- Section 10.7, “Execution Units”

The SEC 3.0 is designed to off-load computationally intensive security functions, such as key generation and exchange, authentication, and bulk encryption from the processor core of the SoC. It is optimized to process all cryptographic algorithms associated with IPsec, IKE, SSL/TLS, iSCSI, SRTP, 802.11i, WiMAX, 3G, A5/3 for GSM and EDGE, and GEA3 for GPRS. The SEC 3.0 is derived from integrated security cores found in other members of the PowerQUICC II and PowerQUICC III families.

The security engine includes eight different execution units (EUs). Where data flows in and out of an EU, each has buffer FIFOs of at least 256 bytes. EU types and features include the following:

- AESU—Advanced Encryption Standard unit
  - Implements the Rijndael symmetric key cipher per U.S. National Institute of Standards and Technology (NIST) Federal Information Processing Standard (FIPS) 197.
  - Modes providing data confidentiality: ECB, CBC, CCM, Counter, GCM, XTS, CBC-RBP, OFB-128, and CFB-128.
  - Modes providing data authentication: CCM, GCM, CMAC (OMAC1), and XCBC-MAC.
  - 128-, 192-, or 256-bit key lengths (only 128-bit keys in XCBC-MAC)
  - ICV (integrity check vector) checking in CCM, GCM, CMAC (OMAC1), and XCBC-MAC mode
  - XOR operations on 2–6 sources for RAID
- AFEU—ARC4 execution unit
  - Implements a stream cipher compatible with the RC4 algorithm
  - 8- to 128-bit programmable key
- CRCU—Cyclical redundancy check unit

- — Implements CRC32C as required for iSCSI header and payload checksums, CRC32 as required for IEEE 802 packets, as well as for programmable CRC polynomials
- — ICV checking
- DEU—Data Encryption Standard execution unit
  - DES, 3DES
  - Two key (K1, K2, K1) or Three Key (K1, K2, K3)
  - ECB, CBC, CFB-64 and OFB-64 modes for both DES and 3DES
- KEU—Kasumi execution unit
  - Implements cipher and authentication modes f8 and f9 used in 3G, A5/3 for GSM and EDGE, and GEA3 for GPRS
  - 128-bit confidentiality key and 128-bit integrity key
  - ICV checking for f9
- MDEU—Message digest execution unit
  - Implements SHA with 160-, 224-, 256-, 384-, or 512-bit message digest (as specified by the FIPS 180-2 standard)
  - Implements MD5 with 128-bit message digest (as specified by RFC 1321)
  - Implements HMAC computation with either message digest algorithm (as specified in RFC 2104 and FIPS-198)
  - Implements SSL MAC computation
  - ICV checking
- PKEU—Public key execution unit that supports the following:
  - RSA and Diffie-Hellman
    - Programmable field size up to 4096 bits
  - Elliptic curve cryptography
    - $F_2$m and $F_p$ modes
    - Programmable field size up to 1023 bits
  - Run time equalization to protect against timing and power attacks
- RNGU—Random number generator unit
  - True Random Number Generator (TRNG)

In addition to the execution units, SEC 3.0 also includes:

- A context switching polychannel, permitting operation of up to four virtual channels, where each channel:
  - Supports a queue of commands (descriptor pointers) to be executed
  - Provides dynamic arbitration for needed crypto-execution units
  - Manages up to two execution units (one ciphering and one hashing), and configures for any required data transfers from one to another
  - Performs flow-control management of buffer FIFOs on the inputs and outputs of execution units

- Supports scatter/gather of input and output data (where the term data is used loosely, and includes keys, context, ICV values, etc.), enabling concatenation of multiple segments of memory when reading or writing data
  - Masters data bursts on 32-byte boundaries to optimize bus throughput
- Master and slave interfaces, with DMA capability
  - 32- or 36-bit address/64-bit data
  - Master interface allows pipelined requests
  - DMA data blocks can start and end on any byte boundary

## 10.1 SEC Architecture Overview

The SEC can act as a master on the internal system bus, allowing it to offload the data movement bottleneck normally associated with slave-only cores. A host processor accesses the SEC through its device drivers using system memory for data storage. The SEC resides in the peripheral memory map of the processor. When an application requires cryptographic functions, it creates descriptors for the SEC which define the functions to be performed and the locations of the data (descriptors are introduced in Section 10.1.1, “Descriptor Overview”, and discussed in detail in Section 10.3, “Descriptors”). With a single 64-bit write, the host processor can enqueue a descriptor pointer in the SEC. The SEC’s bus-mastering capability then enables it to execute the entire cryptographic task, performing reads and writes on system memory as needed.

A block diagram of the SEC internal architecture is shown in Figure 10-1.

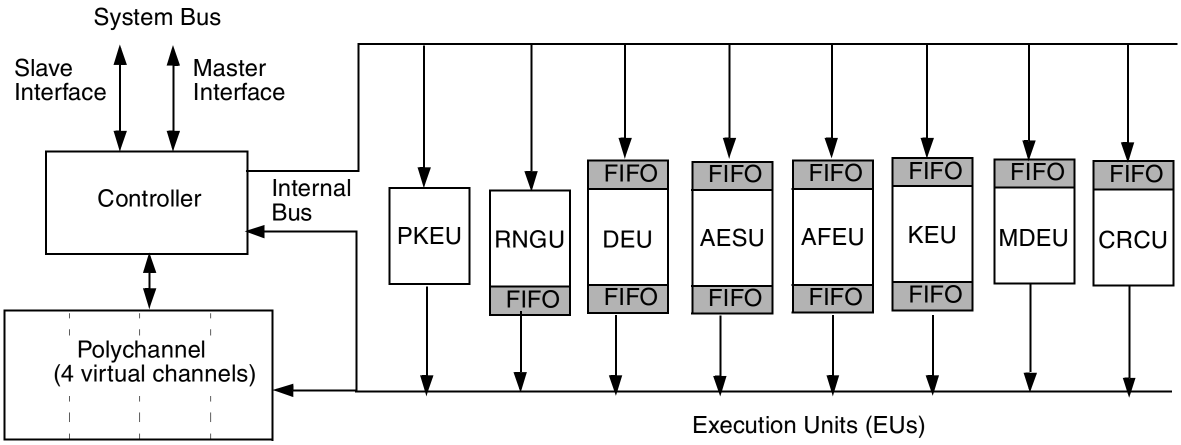


**Figure 10-1. SEC Functional Modules**

The SEC interfaces with the system buses through the controller (the controller is introduced in Section 10.1.3, “Controller Overview”, and discussed in detail in Section 10.5, “Controller”). The slave interface permits an external device to perform 32- or 64-bit writes on any register or FIFO inside the SEC Some locations permit byte writes. Reads may be of any length. Using the master interface, the controller can transfer blocks of 64-bit words between system memory and SEC FIFOs or registers.

A typical SEC operation begins when a host processor writes a descriptor pointer to the fetch FIFO in one of the four SEC virtual channels. This write operation uses the slave interface (where the host is master and SEC is the slave). Following the write, the channel directs the sequence of operations using the master interface (where SEC is master). The channel uses the descriptor pointer to read the descriptor, then decodes the first word of the descriptor to determine the operation to be performed and the crypto-execution unit(s) needed to perform it (the execution units are introduced in Section 10.1.4, "Execution Units (EUs) Overview," and discussed in detail in Section 10.7, "Execution Units"). If necessary, the channel waits for the needed crypto-execution unit(s) to be free. Next, the channel requests the controller to transfer keys, context, and data from memory locations specified in the descriptor be sent to the appropriate execution units. The controller satisfies the requests through its master interface. Data is fed into the execution units through their registers and input FIFOs. The execution units read from their input FIFOs and write processed data to their output FIFOs and registers. The channel requests the controller to write data from the output FIFOs and registers back to system memory.

The channel can signal to the host that it is done with a descriptor by interrupt or by a writeback of the descriptor header into host memory. For more about this signaling, see Section 10.1.2, "Polychannel Overview."

Upon completion of a descriptor, the channel checks the next entry in its fetch FIFO, and (if non-empty) requests a read of the next descriptor.

For most packets, the entire payload is too long to fit in an execution unit's input or output FIFO, so the channel uses a flow control scheme for reading and writing data. The channel directs the controller to read bursts of input as necessary to keep refilling the input FIFO, until the entire payload has been fetched. Similarly, the channel directs the controller to write bursts of output whenever enough accumulates in the execution unit's output FIFO.

The polychannel can process up to four descriptors concurrently by implementing the four virtual channels (the polychannel is introduced in Section 10.1.2, "Polychannel Overview," and discussed in detail in Section 10.4, "Polychannel"). Channels arbitrate for use of execution units, and wait if the needed execution unit is currently reserved by another channel. Each channel has its own FIFO of descriptor pointers to fetch and execute, and its own internal storage. The four channels, however, time-share a single control and datapath unit, and hence they are referred to as virtual channels. A programmable priority scheme allows for round-robin or weighted priorities among these channels.

### 10.1.1 Descriptor Overview

All of the SEC's cryptographic functions are accessible through descriptors. This design facilitates easy use and integration with existing systems and software.

A descriptor specifies cryptographic functions to be performed, and contains reference address pointers to all necessary input data and to the locations where output data is to be written. Some descriptor types perform multiple functions to facilitate particular protocols. A sample descriptor is diagrammed in Table 10-1.Each descriptor contains eight dwords (64 bits each), consisting of the following:

- One dword of header—The header describes the required services and encodes information that indicates which EUs to use and which modes to set. It also indicates whether notification should be sent to the host when the descriptor operation is complete.

- Seven dwords containing pointers and lengths used to locate input or output data. Each pointer can either point directly to the data, or can point to a link table that lists a set of data segments to be concatenated.

**Table 10-1. Example Descriptor**

| Field Name | Value | Description |
|---|---|---|
| Header | 0x2053_1E08_0000_0000 | Example header for IPsec ESP outbound using DES and MD-5 |
| Length0<br>Extent0<br>Pointer0 | 16<br>0<br>(32 or 36-bit pointer) | Number of bytes in authenticate key<br>Unused<br>Pointer to authentication key |
| Length1<br>Extent1<br>Pointer1 | 16<br>0<br>(32 or 36-bit pointer) | Number of bytes in authentication-only data<br>Unused<br>Pointer to authentication-only data |
| Length2<br>Extent2<br>Pointer2 | 8<br>0<br>(32 or 36-bit pointer) | Length of input context (IV)<br>Unused<br>Pointer to input context |
| Length3<br>Extent3<br>Pointer3 | 8<br>0<br>(32 or 36-bit pointer) | Number of bytes in cipher key<br>Unused<br>Pointer to cipher key |
| Length4<br>Extent4<br>Pointer4 | 1500<br>0<br>(32 or 36-bit pointer) | Number of bytes of data to be ciphered<br>Unused<br>Pointer to input data to perform ciphering upon |
| Length5<br>Extent5<br>Pointer5 | 1500<br>12<br>(32 or 36-bit pointer) | Number of bytes of data after ciphering<br>Number of bytes in authentication result (ICV)<br>Pointer to location where cipher output is to be written, followed by ICV |
| Length6<br>Extent6<br>Pointer6 | 8<br>0<br>(32 or 36-bit pointer) | Length of output context (IV)<br>Unused<br>Pointer to location where altered context is to be written |

For more information about descriptors, refer to Section 10.3, “Descriptors.”

## 10.1.2 Polychannel Overview

The polychannel block implements four channels for processing descriptors. Each channel contains the following addressable structures:

- A fetch FIFO, which holds a queue of pointers to descriptors waiting to be processed
- A configuration register, which allows the user a number of options for SEC event signaling
- A status register containing an indication of the last unfulfilled bus request
- A descriptor buffer memory used to store the active descriptor and other temporary data.

Whenever a channel is idle and its fetch FIFO is non-empty, the channel reads the next descriptor pointer from the fetch FIFO. Using this pointer, the channel fetches the descriptor and places it in its descriptor buffer. The channel’s processing of descriptors is described in more detail in Section 10.4.1.1, “Channel Descriptor Processing”

A channel can signal to the host that it is done with a descriptor by interrupt and/or by a writeback of the descriptor header into host memory. In the case of interrupt, there is an option to signal after every descriptor, or only after selected descriptors. In the case of writeback, the value written back is identical to the header that was read, with the exception that a DONE byte is set to 0xFF. The channels' done signaling is described in more detail in Section 10.4.1.3, "Channel Host Notification".

An EU operation can include generating an ICV and then comparing it against a received ICV. The result of the ICV checking can be signalled to the host either by interrupt or by a writeback of the descriptor header. If both are enabled, note that the occurrence of an error interrupt prevents the writeback from occurring. In the case of writeback, the user can opt to do it at end of every descriptor, or only at the end of descriptors that call for ICV checking.

In case of an error condition in a channel or its reserved EUs, the channel issues an interrupt to the host. The channel can be configured to either abort the current descriptor and proceed to the next one, or halt and wait for host intervention.

For more about configuring signaling see Section 10.4.4.1, "Channel Configuration Register (CCR)" and for detail on the writeback fields see Section 10.3.4, "Link Table Format."

Many security protocols involve both encryption and hashing of packet payloads. To accomplish this without requiring two passes through the data, channels can configure data flows through two EUs. In such cases, one EU is designated the "primary EU", and the other as the "secondary EU". The primary EU receives its data from memory through the controller, and the secondary EU receives its data by "snooping" the SEC buses.

There are two types of snooping:

- Input data can be fed to the primary EU and the same input data snooped by the secondary EU. This is called "in-snooping".
- Output data from the primary EU can be snooped by the secondary EU. This is called "out-snooping".

In the SEC, only MDEU and CRCU are used as secondary EUs.

For more information on the polychannel block, refer to Section 10.4, "Polychannel."

### 10.1.3 Controller Overview

The controller manages the master and slave interfaces to the system bus and the internal buses that connect all the various modules. It receives service requests from the host (through the slave interface) and from the channels, and schedules the required data transfers. The system bus interface and access to system memory are critical factors in performance, and the 64-bit master and slave interfaces of the SEC controller enable it to achieve performance unattainable on secondary buses.

The controller enables two modes of operation for the execution units: channel-controlled access and host-controlled access:

- In channel-controlled access (the SEC's normal operating mode), all interactions with EUs are directed by a channel executing a descriptor. The host is involved only in initially supplying the descriptor pointer and in handling results once descriptor processing is complete.

- In host-controlled access (intended primarily for debug purposes), the host moves data in and out of execution units directly through memory-mapped EU registers. No descriptor is involved.

For more information about the controller (including more details about channel-controlled and host-controlled access), refer to Section 10.5, “Controller.”

## 10.1.4 Execution Units (EUs) Overview

“Execution unit” (EU) is the generic term for the functional blocks that perform cryptographic computations. The EUs are compatible with many protocols, and can work together to perform high-level cryptographic tasks. The SEC’s execution units are as follows:

- PKEU for computing asymmetric key operations, including modular exponentiation (and other modular arithmetic functions) or ECC point arithmetic
- DEU for performing block cipher, symmetric key cryptography using DES and 3DES
- AESU for performing the Advanced Encryption Standard algorithm in various modes
- AFEU for performing RC-4 compatible stream cipher symmetric key cryptography
- MDEU for performing security hashing using MD-5, SHA-1, SHA-224, SHA-256, SHA-384 or SHA-512
- KEU for performing 3GPP confidentiality (f8) and integrity (f9) algorithms.
- CRCU for generating cyclical redundancy check values
- RNGU for random number generation

The following sections give an overview of the EUs. Operational details of each EU are given in Section 10.7, “Execution Units.”

### 10.1.4.1 Public Key Execution Unit (PKEU)

The PKEU is capable of performing many advanced mathematical functions to support both RSA and ECC public key cryptographic algorithms. ECC is supported in both $F_2$m (polynomial field) and $F_p$ (prime field) modes.

To assist the host in performing its desired cryptographic functions, the PKEU supports functions with various levels of complexity. For example, at the highest level, the accelerator performs modular exponentiations to support RSA and performs point multiplies to support ECC. At a lower level, the PKEU can perform simple operations such as modular adds and multiplies. For more information about the unit’s operation, refer to Section 10.7.7, “Public Key Execution Units (PKEU).”

#### 10.1.4.1.1 Elliptic Curve Operations

The PKEU has its own data and control units, including a general-purpose register file in the programmable-size arithmetic unit. The field or modulus size can be programmed to any value between 33 bits and 1024 bits in programmable increments of 8, with each programmable value $i$ supporting all actual field sizes from $8i – 7$ to $8i$. The result is hardware supporting a wide range of cryptographic security. Larger field / modulus sizes result in greater security but lower performance.

Compared to RSA, elliptic curve cryptography provides greater security with smaller field sizes. For example, an elliptic curve field size of 160 is roughly equivalent to the security provided by 1024-bit RSA. A field size set to 224 roughly equates to 2048 bits of RSA security.

The PKEU contains routines implementing the atomic functions for elliptic curve processing, including point arithmetic and finite field arithmetic. The point operations (multiplication, addition and doubling) all involve one or more finite field operations which are addition, multiplication, inverse, and squaring. Point add and double each use all four finite field operations. Similarly, point multiplication uses all elliptic curve point operations as well as the finite field operations. All these functions are supported both in prime fields and polynomial fields.

#### 10.1.4.1.2 Modular Exponentiation Operations

The PKEU is also capable of performing integer modulo arithmetic. This arithmetic is an integral part of the RSA public key algorithm; however, it can also play a role in the generation of ECC digital signatures (including ECDSA) and Diffie-Hellman key exchanges.

Modular arithmetic functions supported by the SEC's PKEU include the following (refer to Table 10-67 for a complete list):

- $R^2$ mod N
- $(A \times B)\ R^{-1}$ mod N
- $(A \times B)\ R^{-2}$ mod N
- (A + B) mod N
- (A – B) mod N

In the preceding list, the following notation is used:

- N is the modulus
- A and B are input parameters
- R is $2^{Sz'(N)}$, where Sz'(N) is the bit length of N rounded up to the nearest multiple of 32 (Note: R is referred to as "E" in public key descriptors)

The PKEU can perform modular arithmetic on operands up to 4096 bits in length. The modulus must be larger than or equal to 33 bits (5 bytes), or an error is returned. This is not seen as a limitation since no useful cryptographic applications exist for smaller moduli. The PKEU uses the Montgomery modular multiplication algorithm to perform core functions. The addition and subtraction functions help support known methods of the Chinese Remainder Theorem (CRT) for efficient implementation of the RSA algorithm.

### 10.1.4.2 Data Encryption Standard Execution Unit (DEU)

The DES Execution Unit (DEU) performs bulk data encryption/decryption, in compliance with the Data Encryption Standard algorithm (NIST FIPS 46-3). The DEU can also compute 3DES, an extension of the DES algorithm in which each 64-bit input block is processed three times. The SEC supports 2-key (K1=K3) or 3-key 3DES.

The DEU operates by permuting 64-bit data blocks with a shared 56-bit key and an initialization vector (IV). The SEC supports four modes of operation:

- Electronic Code Book (ECB)
- Cipher Clock Chaining (CBC)
- 64-bit Cipher Feedback Mode (CFB-64)
- 64-bit Output Feedback Mode (OFB-64).

For more information about the unit's operation, refer to Section 10.7.4, "Data Encryption Standard Execution Unit (DEU)."

### 10.1.4.3 Advanced Encryption Standard Execution Unit (AESU)

The AESU is used to accelerate bulk data encryption/decryption in compliance with the Advanced Encryption Standard algorithm Rijndael specified by NIST standard FIPS-197. The AESU executes on 128 bit blocks with a choice of key sizes: 128, 192, or 256 bits.

AES is a symmetric key algorithm, meaning the sender and receiver use the same key for encryption and decryption. The session key and IV are supplied to the AESU module prior to encryption. The processor supplies data to the module that is processed as 128 bit input.

AESU implements the following confidentiality modes from NIST Recommendation 800-38A:

- Electronic Codebook mode (ECB)
- Cipher Block Chaining mode (CBC)
- Output Feedback mode (OFB)
- 128-bit Cipher Feedback mode (CFB-128)
- Counter mode (CTR)

AESU also implements other NIST recommended modes providing authentication (two of which also provide confidentiality):

- Counter with CBC-MAC (CCM) per NIST recommendation 800-38C
- Galois Counter Mode (GCM) per NIST draft recommendation 800-38D
- Cipher-based MAC (CMAC) per NIST recommendation 800-38B.

  Note that CMAC is identical to OMAC1.

AESU modes also implement the following modes not sanctioned by NIST:

- XTS as specified by IEEE P1619 Draft 11
- CBC-RBP
- XCBC-MAC as specified by IETF RFC-3566

In all modes supporting authentication, the AESU hashes data to produce an integrity check vector (ICV). If a reference ICV is supplied to the AESU, it can do a bitwise check of the reference ICV against the one computed by the AESU.

For more information about the unit's operation, refer to Section 10.7.1, "Advanced Encryption Standard Execution Unit (AESU)."

### 10.1.4.4 Arc Four Execution Unit (AFEU)

The AFEU accelerates a bulk encryption algorithm compatible with the RC4 stream cipher from RSA Security, Inc. The algorithm is byte-oriented, meaning the data to be ciphered can be any number of bytes. The AFEU supports key lengths from 8 to 128 bits (in byte increments), providing a wide range of security strengths.

For more information, refer to Section 10.7.2, “ARC4 Execution Unit (AFEU).”

### 10.1.4.5 Message Digest Execution Unit (MDEU)

The MDEU computes a single message digest (or hash or integrity check) value for all the data presented on the input bus. The output size is determined by the specific algorithm, and is typically much smaller than the input size.

The MDEU is designed to support the following hashing algorithms:

- MD5 generates a 128-bit hash, and is specified in RFC 1321.
- SHA-1 is a 160-bit hash function, specified by the NIST FIPS 180-1 standard.
- SHA-224, SHA-256, SHA-384, and SHA-512 are 224-, 256-, 384-, and 512-bit hash functions respectively, specified by the NIST FIPS 180-2 standard.
- The MDEU also supports HMAC computations, as specified by the NIST FIPS-198 standard.

If a digest is supplied to the MDEU, it can do a bitwise check of this supplied digest against the one computed by the MDEU (ICV checking).

For more information about the unit’s operation, refer to Section 10.7.6, “Message Digest Execution Unit (MDEU).”

### 10.1.4.6 Kasumi Execution Unit (KEU)

The KEU (Kasumi Execution Unit) is a functional block capable of encrypting/decrypting and/or performing integrity checks on 64-bit blocks of data using a 128-bit key. The KEU is designed support the following cryptographic algorithms:

- f8 and f9, as defined in the ETSI/SAGE Specification Document 1 for the 3GPP standard
- A5/3 for GSM/EDGE
- GEA3 for GPRS

With the exception of f9, which is an authentication algorithm, KEU implements confidentiality algorithms. For f9, if the KEU is supplied with a MAC value, it is capable of performing a bitwise check of this original MAC against a f9 MAC generated by the KEU (ICV checking).

For more information about the unit’s operation, refer to Section 10.7.5, “Kasumi Execution Unit (KEU).”

### 10.1.4.7 Cyclical Redundancy Check Unit (CRCU)

The CRCU computes a single 32-bit cyclic redundancy code (checksum) from all data presented on the input bus.

The CRC algorithm treats a message stream of bits as coefficients of a massive polynomial and computes the remainder of the modulo two division by an order 32 divisor polynomial. The divisor polynomial is specific to the protocol and chosen to conform to certain mathematical properties to ensure that single bit errors can be detected. Cyclic redundancy codes are used to ensure data integrity over potentially unreliable channels. There are two major CRC protocol algorithms: CRC32 and CRC32C. IEEE 802 defines the CRC32 algorithm, while iSCSI defines the CRC32C algorithm. Both protocols bit swap, byte swap, and then complement the calculated remainder to generate the checksum. The CRCU is designed to support the following check algorithms:

- CRC32 algorithm specified in IEEE 802.1.
- CRC32C algorithm specified in RFC3385.
- A programmable polynomial mode with optional remainder bit mangling is also supported, which can be used to implement proprietary protocols.

The CRCU can perform ICV checking by computing a raw CRC across a message and previously-calculated CRC. Integrity is verified if the result matches the polynomial specific residue.

For more information about the unit’s operation, refer to Section 10.7.3, “Cyclical Redundancy Check Unit (CRCU).”

### 10.1.4.8 Random Number Generator Unit (RNGU)

The RNGU is a functional block that generates 64-bit random numbers and stores them in an output FIFO.

Because many cryptographic algorithms use random numbers as a source for generating a secret value (a nonce), it is desirable to have a private RNG for use by the SEC. The anonymity of each random number must be maintained, as well as the unpredictability of the next random number. The FIPS-140 ‘common criteria’-compliant private RNG allows the system to develop random challenges or random secret keys. The secret key can thus remain hidden from even the high-level application code, providing an added measure of physical security.

For more information about the unit’s operation, refer to Section 10.7.8, “Random Number Generator Unit (RNGU).”

## 10.2 Configuration of Internal Memory Space

Table 10-2 gives the base address map, and shows the blocks of addresses assigned to each SEC sub-block. All address gaps in Table 10-2 are reserved for future use. The 18-bit SEC address bus value is shown. These address values are offsets from the SoC’s base address register (consult the SoC documentation for specific register name).

**Table 10-2. SEC Address Map**

| Byte Address Offset (AD 17–0) | Module | Description | Type | Reference |
|---|---|---|---|---|
| 0x3_1000–0x3_10FF | Controller | Arbiter/controller control register space | Controller | 10.5/10-46 |
| 0x3_1100–0x3_11FF | Channel_1 | Channel 1 | Channels | 10.4/10-32<br>Also see RCA bits in Table 10-21 |
| 0x3_1200–0x3_12FF | Channel_2 | Channel 2 | | |
| 0x3_1300–0x3_13FF | Channel_3 | Channel 3 | | |
| 0x3_1400–0x3_14FF | Channel_4 | Channel 4 | | |
| 0x3_1500–0x3_16FF | PolyChn | PolyChannel | | 10.4.3/10-35 |
| 0x3_1BF8 | Controller | IP block revision register | Read only | 10.5.4.5/10-54 |
| 0x3_2000–0x3_2FFF | DEU | DES/3DES execution unit | Crypto EU | 10.7.4/10-108 |
| 0x3_4000–0x3_4FFF | AESU | AES execution unit | | 10.7.1/10-57 |
| 0x3_6000–0x3_6FFF | MDEU | Message digest execution unit | | 10.7.6/10-132 |
| 0x3_8000–0x3_8FFF | AFEU | Arc Four execution unit | | 10.7.2/10-88 |
| 0x3_A000–0x3_AFFF | RNGU | Random number generator unit | | 10.7.8/10-155 |
| 0x3_C000–0x3_CFFF | PKEU | Public key execution unit | | 10.7.7/10-146 |
| 0x3_E000–0x3_EFFF | KEU | Kasumi execution unit | | 10.7.5/10-117 |
| 0x3_F000–0x3_FFFF | CRCU | Cyclical Redundancy Check Unit | | 10.7.3/10-98 |

Table 10-3 shows the detailed system address map showing all functional registers.

All SEC registers are allocated 8 bytes (one dword), and the addresses listed in the table are all at 8-byte boundaries (addresses end in 0 or 8). It is possible, however, to access the registers by 4-byte words, or in some cases by byte. The "Write by" column in Table 10-3 distinguishes these cases. The column entries are interpreted as follows:

- Byte: This register can be written by byte (using any address), by word (using an address ending in 0, 4, 8, or C), or by dword (using an address ending in 0 or 8).
- Word: This register can be written by dword (using an address ending in 0 or 8), or by word (using an address ending in 0, 4, 8, or C), but not by byte.

Reads can always be done by byte, word, or dword.

**Table 10-3. SEC Address Map**

| Byte Address Offset (AD 17–0) | Module | Register | Access | Write by | Reference |
|---|---|---|---|---|---|
| 0x3_1008 | Controller | Interrupt enable | R/W | byte[1] | 10.5.4.2/1010-50 |
| 0x3_1010 | | Interrupt status | R | — | 10.5.4.2/1010-53 |
| 0x3_1018 | | Interrupt clear | R/W | byte | 10.5.4.3/1010-54 |
| 0x3_1020 | | Identification | R | — | 10.5.4.4/1010-54 |
| 0x3_1028 | | EU assignment status | R | — | 10.5.4.1/1010-50 |
| 0x3_1030 | | Master control | R/W | byte | 10.5.4.6/1010-55 |
| 0x3_1108 | Channel_1 | Configuration register | R/W | word | 10.4.4.1/1010-37 |
| 0x3_1110 | | Pointer status | R/W | word | 10.4.4.2/1010-41 |
| 0x3_1140 | | Current descriptor pointer | R | — | 10.4.4.3/1010-43 |
| 0x3_1148 | | Fetch FIFO | W | word | 10.4.4.4/1010-44 |
| 0x3_1180–0x3_11BF | | Descriptor buffer | R | — | 10.4.5.1/1010-45 |
| 0x3_11C0–0x3_11DF | | Gather Link Table | R | — | 10.4.5.2/1010-45 |
| 0x3_11E0–0x3_11FF | | Scatter Link Table | R | — | 10.4.5.2/1010-45 |
| 0x3_1208 | Channel_2 | Configuration register | R/W | word | 10.4.4.1/1010-37 |
| 0x3_1210 | | Pointer status | R/W | word | 10.4.4.2/1010-41 |
| 0x3_1240 | | Current descriptor pointer | R | — | 10.4.4.3/1010-43 |
| 0x3_1248 | | Fetch FIFO | W | word | 10.4.4.4/1010-44 |
| 0x3_1280–0x3_12BF | | Descriptor buffer | R | — | 10.4.5.1/1010-45 |
| 0x3_12C0–0x3_12DF | | Gather Link Table | R | — | 10.4.5.2/1010-45 |
| 0x3_12E0–0x3_12FF | | Scatter Link Table | R | — | 10.4.5.2/1010-45 |
| 0x3_1308 | Channel_3 | Configuration register | R/W | word | 10.4.4.1/1010-37 |
| 0x3_1310 | | Pointer status | R/W | word | 10.4.4.2/1010-41 |
| 0x3_1340 | | Current descriptor pointer | R | — | 10.4.4.3/1010-43 |
| 0x3_1348 | | Fetch FIFO | W | word | 10.4.4.4/1010-44 |
| 0x3_1380–0x3_13BF | | Descriptor buffer | R | — | 10.4.5.1/1010-45 |
| 0x3_13C0–0x3_13DF | | Gather Link Table | R | — | 10.4.5.2/1010-45 |
| 0x3_13E0–0x3_13FF | | Scatter Link Table | R | — | 10.4.5.2/1010-45 |

Table 10-3. SEC Address Map (continued)

| Byte Address Offset (AD 17–0) | Module | Register | Access | Write by | Reference |
|---|---|---|---|---|---|
| 0x3_1408 | Channel_4 | Configuration register | R/W | word | 10.4.4.1/1010-37 |
| 0x3_1410 | | Pointer status | R/W | word | 10.4.4.2/1010-41 |
| 0x3_1440 | | Current descriptor pointer | R | — | 10.4.4.3/1010-43 |
| 0x3_1448 | | Fetch FIFO | W | word | 10.4.4.4/1010-44 |
| 0x3_1480–0x3_14BF | | Descriptor buffer | R | — | 10.4.5.1/1010-45 |
| 0x3_14C0–0x3_14DF | | Gather Link Table | R | — | 10.4.5.2/1010-45 |
| 0x3_14E0–0x3_14FF | | Scatter Link Table | R | — | 10.4.5.2/1010-45 |
| 0x3_1500 | Poly-Channel | Fetch FIFO Enqueue Count | R/W | word | 10.4.3.1/1010-35 |
| 0x3_1508 | | Descriptor Finished Count | R/W | word | 10.4.3.1/1010-36 |
| 0x3_1510 | | Data Bytes In Count | R/W | word | 10.4.3.1/1010-36 |
| 0x3_1518 | | Data Bytes Out Count | R/W | word | 10.4.3.1/1010-37 |
| 0x3_1BF8 | Controller | IP block revision | R | — | 10.5.4.5/1010-54 |
| 0x3_2000 | DEU | Mode register | R/W | word | 10.7.4.1/1010-109 |
| 0x3_2008 | | Key size register | R/W | word | 10.7.4.2/1010-110 |
| 0x3_2010 | | Data size register | R/W | word | 10.7.4.3/1010-110 |
| 0x3_2018 | | Reset control register | R/W | word | 10.7.4.4/1010-111 |
| 0x3_2028 | | Status register | R | — | 10.7.4.5/1010-112 |
| 0x3_2030 | | Interrupt status register | R/W | word | 10.7.4.6/1010-113 |
| 0x3_2038 | | Interrupt mask register | R/W | word | 10.7.4.7/1010-115 |
| 0x3_2050 | | EU-Go | W | word | 10.7.4.8/1010-116 |
| 0x3_2100 | | IV register | R/W | word | 10.7.4.9/1010-117 |
| 0x3_2400 | | Key 1 register | W | byte | 10.7.4.10/1010-117 |
| 0x3_2408 | | Key 2 register | W | byte | 10.7.4.10/1010-117 |
| 0x3_2410 | | Key 3 register | W | byte | 10.7.4.10/1010-117 |
| 0x3_2800–0x3_2FFF | | Input FIFO / Output FIFO | R/W[2] | byte | 10.7.4.11/1010-117 |

**Table 10-3. SEC Address Map (continued)**

| Byte Address Offset (AD 17–0) | Module | Register | Access | Write by | Reference |
|---|---|---|---|---|---|
| 0x3_4000 | AESU | Mode register | R/W | word | 10.7.1.2/1010-58 |
| 0x3_4008 | | Key size register | R/W | word | 10.7.1.3/1010-61 |
| 0x3_4010 | | Data size register | R/W | word | 10.7.1.4/1010-61 |
| 0x3_4018 | | Reset control register | R/W | word | 10.7.1.5/1010-62 |
| 0x3_4028 | | Status register | R | — | 10.7.1.6/1010-62 |
| 0x3_4030 | | Interrupt status register | R/W | word | 10.7.1.7/1010-64 |
| 0x3_4038 | | Interrupt mask register | R/W | word | 10.7.1.8/1010-66 |
| 0x3_4040 | | ICV size register | R/W | word | 10.7.1.9/1010-67 |
| 0x3_4050 | | End of message register | W | word | 10.7.1.10/1010-68 |
| 0x3_4100–0x3_415F | | Context | R/W | byte | 10.7.1.11/1010-68 |
| 0x3_4400–0x3_441F | | Key registers | R/W | byte | 10.7.1.12/1010-87 |
| 0x3_4800–0x3_4FFF | | Input FIFO / Output FIFO | R/W [1] | byte | 10.7.1.12/1010-88 |
| 0x3_6000 | MDEU | Mode register | R/W | word | 10.7.6.2/1010-132 |
| 0x3_6008 | | Key size register | R/W | word | 10.7.6.4/1010-136 |
| 0x3_6010 | | Data size register | R/W | word | 10.7.6.5/1010-137 |
| 0x3_6018 | | Reset control register | R/W | word | 10.7.6.6/1010-137 |
| 0x3_6028 | | Status register | R | — | 10.7.6.7/1010-138 |
| 0x3_6030 | | Interrupt status register | R/W | word | 10.7.6.8/1010-139 |
| 0x3_6038 | | Interrupt mask register | R/W | word | 10.7.6.9/1010-141 |
| 0x3_6040 | | ICV size register | W | word | 10.7.6.10/1010-142 |
| 0x3_6050 | | End of message reigster | W | word | 10.7.6.11/1010-143 |
| 0x3_6100–0x3_6147 | | Context registers | R/W | byte | 10.7.6.12/1010-143 |
| 0x3_6400–0x3_647F | | Key registers | W | byte | 10.7.6.13/1010-146 |
| 0x3_6800–0x3_6FFF | | Input FIFO | W [1] | byte | 10.7.6.14/1010-146 |

Table 10-3. SEC Address Map (continued)

| Byte Address Offset (AD 17–0) | Module | Register | Access | Write by | Reference |
|---|---|---|---|---|---|
| 0x3_8000 | AFEU | Mode register | R/W | word | 10.7.2.1/1010-89 |
| 0x3_8008 | | Key size register | R/W | word | 10.7.2.2/1010-89 |
| 0x3_8010 | | Data size register | R/W | word | 10.7.2.3/1010-90 |
| 0x3_8018 | | Reset control register | R/W | word | 10.7.2.4/1010-91 |
| 0x3_8028 | | Status register | R | — | 10.7.2.5/1010-91 |
| 0x3_8030 | | Interrupt status register | R/W | word | 10.7.2.6/1010-92 |
| 0x3_8038 | | Interrupt mask register | R/W | word | 10.7.2.7/1010-94 |
| 0x3_8050 | | End of message register | W | word | 10.7.2.8/1010-96 |
| 0x3_8100–0x3_81FF | | Context memory | R/W | byte | 10.7.2.9/1010-96 |
| 0x3_8200 | | Context memory pointers | R/W | byte | 10.7.2.9/1010-96 |
| 0x3_8400–0x3_840F | | Key registers | W | byte | 10.7.2.10/1010-97 |
| 0x3_8800–0x3_8FFF<br>(3_8E00) | | Input FIFO / Output FIFO<br>(special context address) | R/W [1] | byte | 10.7.2.10/1010-97 |
| 0x3_A000 | RNGU | Mode register | R/W | word | 10.7.8.1/1010-156 |
| 0x3_A010 | | Data size register | R/W | word | 10.7.8.2/1010-156 |
| 0x3_A018 | | Reset control register | R/W | word | 10.7.8.3/1010-156 |
| 0x3_A028 | | Status register | R | — | 10.7.8.4/1010-157 |
| 0x3_A030 | | Interrupt status register | R/W | word | 10.7.8.5/1010-158 |
| 0x3_A038 | | Interrupt mask register | R/W | word | 10.7.8.6/1010-159 |
| 0x3_A050 | | End of message register | W | word | 10.7.8.7/1010-160 |
| 0x3_A400–0x3_A43F | | Entropy registers | W | word | 10.7.8.8/1010-161 |
| 0x3_A800–0x3_AFFF | | Output FIFO | R [1] | — | 10.7.8.8/1010-161 |

Table 10-3. SEC Address Map (continued)

| Byte Address Offset (AD 17–0) | Module | Register | Access | Write by | Reference |
|---|---|---|---|---|---|
| 0x3_C000 | PKEU | Mode register | R/W | word | 10.7.7.1/1010-147 |
| 0x3_C008 | | Key size register | R/W | word | 10.7.7.2/1010-147 |
| 0x3_C010 | | Data size register | R/W | word | 10.7.7.4/1010-149 |
| 0x3_C018 | | Reset control register | R/W | word | 10.7.7.5/1010-149 |
| 0x3_C028 | | Status register | R | — | 10.7.7.6/1010-150 |
| 0x3_C030 | | Interrupt status register | R/W | word | 10.7.7.7/1010-151 |
| 0x3_C038 | | Interrupt mask register | R/W | word | 10.7.7.8/1010-153 |
| 0x3_C040 | | ABSize | R/W | word | 10.7.7.3/1010-148 |
| 0x3_C050 | | End of message register | W | word | 10.7.7.9/1010-154 |
| 0x3_C200–0x3_C27F | | Parameter memory A0 | R/W | byte | 10.7.7.10/1010-154 |
| 0x3_C280–0x3_C2FF | | Parameter memory A1 | R/W | byte | |
| 0x3_C300–0x3_C37F | | Parameter memory A2 | R/W | byte | |
| 0x3_C380–0x3_C3FF | | Parameter memory A3 | R/W | byte | |
| 0x3_C400–0x3_C47F | | Parameter memory B0 | R/W | byte | |
| 0x3_C480–0x3_C4FF | | Parameter memory B1 | R/W | byte | |
| 0x3_C500–0x3_C57F | | Parameter memory B2 | R/W | byte | |
| 0x3_C580–0x3_C5FF | | Parameter memory B3 | R/W | byte | |
| 0x3_C800–0x3_C9FF | | Parameter memory N | R/W | byte | |
| 0x3_CA00–0x3_CBFF | | Parameter memory E | W | byte | |

### Table 10-3. SEC Address Map (continued)

| Byte Address Offset (AD 17–0) | Module | Register | Access | Write by | Reference |
|---|---|---|---|---|---|
| 0x3_E000 | KEU | Mode register | R/W | word | 10.7.5.1/1010-118 |
| 0x3_E008 | | Key size register | R/W | word | 10.7.5.2/1010-119 |
| 0x3_E010 | | Data size register | R/W | word | 10.7.5.3/1010-120 |
| 0x3_E018 | | Reset control register | R/W | word | 10.7.5.4/1010-121 |
| 0x3_E028 | | Status register | R | — | 10.7.5.5/1010-122 |
| 0x3_E030 | | Interrupt Status register | R/W | word | 10.7.5.6/1010-123 |
| 0x3_E038 | | Interrupt Mask register | R/W | word | 10.7.5.7/1010-125 |
| 0x3_E048 | | Data out register (f9 MAC) | R | — | 10.7.5.8/1010-127 |
| 0x3_E050 | | End of message register | W | word | 10.7.5.9/1010-127 |
| 0x3_E100 | | IV_1 register | R/W | byte | 10.7.5.10/1010-128 |
| 0x3_E108 | | ICV_In register | R/W | byte | 10.7.5.11/1010-129 |
| 0x3_E110 | | IV_2 register (FRESH) | R/W | byte | 10.7.5.12/1010-129 |
| 0x3_E118 | | Context_1 register | R/W | byte | 10.7.5.13/1010-129 |
| 0x3_E120 | | Context_2 register | R/W | byte | 10.7.5.13/1010-129 |
| 0x3_E128 | | Context_3 register | R/W | byte | 10.7.5.13/1010-129 |
| 0x3_E130 | | Context_4 register | R/W | byte | 10.7.5.13/1010-129 |
| 0x3_E138 | | Context_5 register | R/W | byte | 10.7.5.13/1010-129 |
| 0x3_E140 | | Context_6 register | R/W | byte | 10.7.5.13/1010-129 |
| 0x3_E400 | | Key data register_1 (CK-high) | R/W | byte | 10.7.5.14/1010-130 |
| 0x3_E408 | | Key data register_2 (CK-low) | R/W | byte | 10.7.5.14/1010-130 |
| 0x3_E410 | | Key dataregister_3 (IK-high) | R/W | byte | 10.7.5.15/1010-130 |
| 0x3_E418 | | Key data register_4 (IK-low) | R/W | byte | 10.7.5.15/1010-130 |
| 0x3_E800–0x3_EFFF | | Input FIFO / Output FIFO | R/W [1] | byte | 10.7.5.16/1010-131 |

Table 10-3. SEC Address Map (continued)

| Byte Address Offset (AD 17–0) | Module | Register | Access | Write by | Reference |
|---|---|---|---|---|---|
| 0x3_F000 | CRCU | Mode register | R/W | word | 10.7.3.2/1010-98 |
| 0x3_F008 | | Key size register | R/W | word | 10.7.3.3/1010-99 |
| 0x3_F010 | | Data size register | R/W | word | 10.7.3.4/1010-100 |
| 0x3_F018 | | Reset control register | R/W | word | 10.7.3.5/1010-100 |
| 0x3_F020 | | Control | R/W | word | 10.7.3.6/1010-101 |
| 0x3_F028 | | Status register | R | — | 10.7.3.7/1010-101 |
| 0x3_F030 | | Interrupt status register | R/W | word | 10.7.3.8/1010-102 |
| 0x3_F038 | | Interrupt mask register | R/W | word | 10.7.3.9/1010-104 |
| 0x3_F040 | | ICV size register | R/W | word | 10.7.1.9/1010-67 |
| 0x3_F050 | | End of message register | W | word | 10.7.3.11/1010-106 |
| 0x3_F108 | | Context register | R/W | byte | 10.7.3.12/1010-106 |
| 0x3_F400 | | Key register | R/W | byte | 10.7.3.13/1010-108 |
| 0x3_F800–0x3_FFFF | | Input FIFO | W [1] | byte | 10.7.3.14/1010-108 |

[1] Byte accessibility is controlled by internal logic, particularly at FIFOs, to prevent unintended overwrites of partial words during writes, and to prevent unintended duplicate reads of partial data during reads. In addition, these bytes must be presented on the correct byte lanes for the intended destination.

[2] For the EU FIFOs, write operations anywhere in the address range enqueue to the input FIFO, and read operations anywhere in the address range dequeue from the output FIFO. See the referenced section for more detailed information.

## 10.3 Descriptors

The host processor maintains a record of current secure sessions and the corresponding keys and contexts of those sessions. Once the host has determined that a security operation is required, it creates a “descriptor” containing all the information the SEC needs to perform the security operation. The host creates the descriptor in main memory, then writes a pointer to the descriptor into the fetch FIFO of one of the SEC channels. The channel uses this pointer to read the descriptor into its descriptor buffer. Once it obtains the descriptor, the SEC uses its bus mastering capability to obtain inputs and write results, thus offloading data movement and encryption operations from the host processor.

Descriptors are only used in channel-controlled accesses to SEC, and not in host-controlled accesses. For more information about host-controlled access, see Section 10.5.1.1, “Host-Controlled Access”.

## 10.3.1 Descriptor Structure

SEC descriptors are designed so that a single descriptor supports the cryptographic computation of a single packet. SEC descriptors have a fixed length of 64 bytes, that is, eight 64-bit words (referred to as dwords). A descriptor consists of one header dword and seven "pointer dwords," as seen in Figure 10-2.

| | 0 15 | 16 | 17 23 | 2427 | 28 31 | 32 63 |
|---|---|---|---|---|---|---|
| Header Dword | Descriptor Control | | | | | Descriptor Feedback |
| Pointer Dword 0 | Length0 | J0 | Extent0 | — | Eptr0 | Pointer0 |
| Pointer Dword 1 | Length1 | J1 | Extent1 | — | Eptr1 | Pointer1 |
| Pointer Dword 2 | Length2 | J2 | Extent2 | — | Eptr2 | Pointer2 |
| Pointer Dword 3 | Length3 | J3 | Extent3 | — | Eptr3 | Pointer3 |
| Pointer Dword 4 | Length4 | J4 | Extent4 | — | Eptr4 | Pointer4 |
| Pointer Dword 5 | Length5 | J5 | Extent5 | — | Eptr5 | Pointer5 |
| Pointer Dword 6 | Length6 | J6 | Extent6 | — | Eptr6 | Pointer6 |

**Figure 10-2. Descriptor Format**

As shown in Figure 10-2, the first and second halves of the header dword are denoted as descriptor control and descriptor feedback fields, respectively. The descriptor control field of the header dword specifies the security operation to be performed, the execution unit(s) needed, and the modes for each execution unit. The descriptor feedback field is written to by the security engine upon completion of descriptor processing, when the "channel done writeback" feature is enabled. Further details about the header dword may be found in Section 10.3.2, "Descriptor Format: Header Dword."

The pointer dwords, all of which have the same format, contain pointer and length information for locating input or output parcels (such as keys, context, or text data). The large number of pointers provided in the descriptor allows for multi-algorithm operations that require fetching of multiple keys, as well as fetch and return of contexts. Any pointer dword that is not needed may be assigned a length of zero. Further details about the pointer dwords may be found in Section 10.3.3, "Descriptor Format: Pointer Dwords."

SEC descriptors include scatter/gather capability, which means that each pointer in a descriptor can be either a direct pointer to a contiguous parcel of data, or a pointer to a "link table" which is a list of pointers and lengths used to assemble the parcel. When a link table is used to read input data, this is referred to as a "gather" operation; when used to write output data, it is referred to as a "scatter" operation. Further details about scatter/gather capability may be found in Section 10.3.4, "Link Table Format."

## 10.3.2 Descriptor Format: Header Dword

Descriptors are created by the host to guide the SEC through required cryptographic operations. The header dword provides the primary indication of the operations to be performed, the mode for each operation, and internal addressing used by the controller and channel for internal data movement. The fields that must be supplied to SEC are shown in the "Field" rows of Figure 10-3 and described in Table 10-4. The SEC device drivers allow the host to create proper headers for each cryptographic operation.

SEC processing of a descriptor sometimes includes writing the original header dword back to system memory with certain fields modified. The modified fields are shown in the “Writeback” rows of Figure 10-3 and described in Table 10-5.

| | 0 3 | 4 7 | 8 11 | 12 15 | 16 23 | 24 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|
| Field | OP_0 | | | OP_1 | | DESC_TYPE | — | DIR | DN |
| | EU_SEL0 | MODE0 | | EU_SEL1 | MODE1 | | | | |
| Writeback | DONE | | — | | | | | | |

| | 32 34 | 35 36 | 37 42 | 43 44 | 45 47 | 48 63 |
|---|---|---|---|---|---|---|
| Field | — | | | | | |
| Writeback | — | ICCR0 | — | ICCR1 | — | ID_TAG |

**Figure 10-3. Header Dword**

**Table 10-4. Header Dword Bit Definitions**

| Bits | Name | Description |
|---|---|---|
| | OP_0: | |
| 0–3 | EU_SEL0 | Primary execution unit select: See Section 10.3.2.1, “Selecting Execution Units—EU_SEL0 and EU_SEL1,” for possible values. |
| 4–11 | MODE0 | Primary mode: Mode data used to program the primary EU. The mode data is specific to the chosen EU. This field is passed directly to bits 56–63 of the mode register in the selected EU. Refer to the EU-specific mode register sections (Section 10.7.1.2, “AESU Mode Register,” Section 10.7.2.1, “AFEU Mode Register,” Section 10.7.3.2, “CRCU Mode Register,” Section 10.7.4.1, “DEU Mode Register,” Section 10.7.5.1, “KEU Mode Register (KEUMR),” Section 10.7.6.2, “MDEU Mode Register,” Section 10.7.7.1, “PKEU Mode Register,” and Section 10.7.8.1, “RNGU Mode Register”) for further info. Any bits of any use in any mode register beyond bits 56–63 are under control of the channel and not the MODE0 field. |
| | OP_1: | |
| 12–15 | EU_SEL1 | Secondary EU select: See Section 10.3.2.1, “Selecting Execution Units—EU_SEL0 and EU_SEL1,” for possible values. |
| 16–23 | MODE1 | Secondary mode: Mode data used to program the primary EU. The mode data is specific to the chosen EU. This field is passed directly to bits 56–63 of the mode register in the selected EU. Refer to the EU-specific mode register sections (sections Section 10.7.3.2, “CRCU Mode Register,” and Section 10.7.6.2, “MDEU Mode Register”) for further info. |
| 24–28 | DESC_TYPE | Descriptor Type: This, along with the DIR field, determines the sequence of actions to be performed by the channel and selected EUs using the blocks of data listed in the rest of the descriptor. The attributes determined include the direction of data flow for each data block, which EU (primary or secondary) is accessed, what snooping options are used, and address offsets for internal EU accesses.<br>See Section 10.3.2.2, “Selecting Descriptor Type—DESC_TYPE,” for possible values. |
| 29 | — | Reserved |

Table 10-4. Header Dword Bit Definitions (continued)

| Bits | Name | Description |
|---|---|---|
| 30 | DIR | Direction: direction of overall data flow:<br>0 Outbound<br>1 Inbound<br>This, along with the DESC_TYPE field, helps determine the sequence of actions to be performed by the channel and selected EUs. |
| 31 | DN | Done notification:<br>0 No done notification.<br>1 Signal “done” to the host on completion of this descriptor.<br>This enables done notification if the NT bit is set in the channel configuration register (see Table 10-11). The done notification can take the form of an interrupt, a writeback in the DONE field of this header dword (see Table 10-5), or both, depending upon the states of the CDIE (channel done interrupt enable) and CDWE (channel done writeback enable) bits in the channel configuration register. |

Table 10-5. Header Dword Writeback Bit Definitions

| Bits | Name | Description |
|---|---|---|
| 0–7 | DONE | When “channel done writeback” is enabled, then at the completion of descriptor processing this byte is written with the value 0xFF. “Channel done writeback” is enabled by programming the CDWE, NT, and CDIE fields in the channel configuration register (see Table 10-11). |
| 8–34 | — | Reserved |
| 35–36 | ICCR0 | Integrity check comparison result from primary: These bits are supplied by the primary EU when descriptor processing is complete.<br>00 No integrity check comparison was performed.<br>01 The integrity check comparison passed.<br>10 The integrity check comparison failed.<br>11 Reserved |
| 37–42 | — | Reserved |
| 43–44 | ICCR1 | Integrity check comparison result from secondary: These bits are supplied by the secondary EU (if any) when descriptor processing is complete.<br>00 No integrity check comparison was performed.<br>01 The integrity check comparison passed.<br>10 The integrity check comparison failed.<br>11 Reserved |
| 45–47 | — | Reserved |
| 48–63 | ID_TAG | Identification Tag. This value is copied from the ID_TAG field written by the host into the fetch FIFO (see Section 10.4.4.4, “Fetch FIFO Enqueue Register (FFER)”). |

#### 10.3.2.1 Selecting Execution Units—EU_SEL0 and EU_SEL1

Table 10-6 shows the values for EU_SEL0 and EU_SEL1 in the descriptor header. The following rules govern the choices for these fields:

1. EU_SEL0 values of “No EU selected” or “Reserved” result in an “Unrecognized header” error condition during processing of the descriptor header.

2. The only valid choices for EU_SEL1 are “No EU selected”, CRCU, or MDEU. Any other choice results in an “Unrecognized header” error condition.
3. If EU_SEL1 is CRCU or MDEU, then EU_SEL0 must be DEU, AESU, AFEU, or KEU. All other values of EU_SEL0 result in an “Unrecognized header” error condition.

**Table 10-6. EU_SEL0 and EU_SEL1 Values**

| Value (binary) | Selected EU |
|---|---|
| 0000 | No EU selected |
| 0001 | AFEU |
| 0010 | DEU |
| 0011 | MDEU-A |
| 1011 | MDEU-B |
| 0100 | RNGU |
| 0101 | PKEU |
| 0110 | AESU |
| 0111 | KEU |
| 1000 | CRCU |
| others | Reserved |
| 1111 | Reserved for header writeback |

The designators MDEU-A and MDEU-B both refer to the same physical MDEU. If MDEU-B is selected, then the channel configures MDEU to perform SHA-224, SHA-256, SHA-384, and SHA-512. If MDEU-A is selected, then the channel configures MDEU to perform SHA-160, SHA-224, SHA-256, or MD5. This configuration is achieved automatically; the channel sets bit 51 of the MDEU mode register as it inserts the MODE0 (or MODE1) value into the MDEU mode register. For further information see Section 10.7.6.2, “MDEU Mode Register.”

### 10.3.2.2 Selecting Descriptor Type—DESC_TYPE

Table 10-7 shows the permissible values for the DESC_TYPE field in the descriptor header. Descriptor types from the SEC1.0, which have “0” in the last bit, are listed first, followed by new SEC 2.x/3.x types, which have “1” in the last bit.

**Table 10-7. Descriptor Types**

| Value (binary) | Descriptor Type | Notes |
|---|---|---|
| 0000_0 | aesu_ctr_nonsnoop | AESU CTR nonsnooping |
| 0001_0 | common_nonsnoop | Common, nonsnooping, non-PKEU, non-AFEU |
| 0010_0 | hmac_snoop_no_afeu | Snooping, HMAC, non-AFEU |
| 0011_0 | — | Reserved |

Table 10-7. Descriptor Types (continued)

| Value (binary) | Descriptor Type | Notes |
|---|---|---|
| 0100_0 | — | Reserved |
| 0101_0 | common_nonsnoop_afeu | Common, nonsnooping, AFEU |
| 0110_ 0 | — | Reserved |
| 0111_0 | — | Reserved |
| 1000_0 | pkeu_mm | PKEU-Montgomery Multiplication |
| 1001_0 | — | Reserved |
| 1010_0 | — | Reserved |
| 1011_0 | — | Reserved |
| 1100_0 | hmac_snoop_aesu_ctr | AESU CTR hmac snooping [2] |
| 1101_0 | — | Reserved |
| 1110_0 | — | Reserved |
| 1111_0 | — | Reserved |
| 0000_1 | ipsec_esp | IPsec ESP mode encryption and hashing |
| 0001_1 | 802.11i_aes_ccmp | CCMP encryption and hashing, suitable for 802.11i |
| 0010_1 | srtp | SRTP encryption and hashing |
| 0011_1 | pkeu_build | pkeu_build Elliptic Curve Cryptography |
| 0100_1 | pkeu_ptmul | pkeu_ptmul Elliptic Curve Cryptography |
| 0101_1 | pkeu_ptadd_dbl | pkeu_ptadd_dbl Elliptic Curve Cryptography |
| 0110_1 | — | Reserved |
| 0111_1 | — | Reserved |
| 1000_1 | tls_ssl_block | TLS/SSL generic block cipher |
| 1001_1 | tls_ssl_stream | TLS/SSL generic stream cipher |
| 1010_1 | raid_xor | XOR 2-6 sources together |
| 1011_1 | ipsec_aes_gcm | IPsec ESP mode using AES GCM encryption and hashing |
| 1100_1 | dbl_crc | Do two CRC operations |
| others | — | Reserved |

[1] Type 0000_0 is for AES-CTR operations. Type 0001_0 also supports AES-CTR, however to use AES-CTR with 0001_0, the user must prepend zeros to the AES-Context before loading the AES Context Registers.

[2] Type 1100_0 is for AES-CTR operations with HMAC. Type 0010_0 also supports AES-CTR with HMAC, however to use AES-CTR with 0010_0, the user must prepend zeros to the AES-Context before loading the AES Context Registers.

For more about descriptor types and the data used for each type, see Section 10.3.5, "Descriptor Types."

### 10.3.3 Descriptor Format: Pointer Dwords

The descriptor contains seven “pointer dwords” which define where in memory the SEC should access its input and output parcels. The pointer dwords are numbered 0 to 6 as shown in Figure 10-2. The channel determines how it uses each of the pointer dwords based on the descriptor type and direction fields in the header (see Table 10-4).

The pointer dword bit fields as shown in Figure 10-4, and are described in Table 10-8.

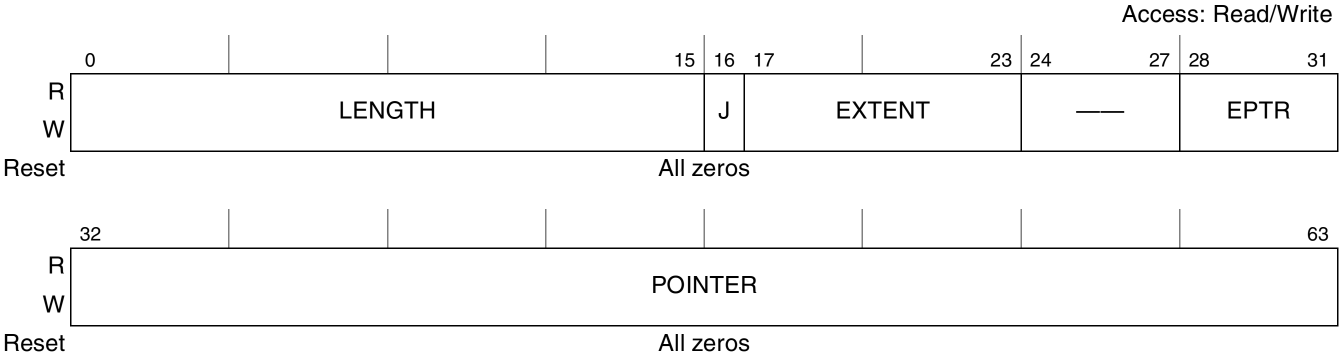


**Figure 10-4. Pointer Dword**

**Table 10-8. Pointer Dword Field Definitions**

| Bits | Name | Description |
|---|---|---|
| 0-15 | LENGTH | Length: A number of bytes in the range 0 to 65535. The use of this field depends on the descriptor type and direction fields in the header dword. A value of zero may cause the channel to skip this dword. |
| 16 | J | Jump: Determines whether to “jump” to a link table whenever the POINTER field in this same dword is used.<br>0 The POINTER field points to data.<br>1 The POINTER field points to a link table, and scatter/gather is enabled. |
| 17-23 | EXTENT | Extent: A number of bytes in the range 0 to 127. The use of this field depends on the “Descriptor Type” and “Direction” fields in the header dword. |
| 24-27 | — | Reserved |
| 28-31 | EPTR | Extended Pointer: Concatenated as the top 4 bits of the pointer when EAE is high (see the EAE bit in Table 10-11). |
| 32-63 | POINTER | Pointer: A memory address. |

On occasion, a descriptor field may not be applicable to the requested service. With seven pointer dwords, it is possible that not all these dwords are required to specify the input and output parameters (for instance, some operations do not require context.) Wherever a particular field is not used, it should be set to zero.

The channel proceeds linearly through the descriptor, fetching LENGTH data beginning at location POINTER. If the EAE (extend address enable) bit is set in the channel configuration register (see Table 10-11), then the four EPTR bits are concatenated with the POINTER field to form a 36-bit pointer address.

If all the data of LENGTH is found contiguously beginning at POINTER, then the Jump bit is not set. Otherwise, POINTER indicates the location of a link table (scatter-gather list). For more details, see Section 10.3.4, “Link Table Format”.

LENGTH and EXTENT fields normally specify the sizes of parcels: often (but not always) the size of the parcel located at the address contained in the matching POINTER field[1]. However, in some cases the POINTER field is zero, and the LENGTH and/or EXTENT fields simply specify values to be written to an EU. The specific use of these fields in each channel depends on the descriptor type and direction fields in the descriptor’s header dword (see Table 10-4).

The RAID-XOR descriptor type does not support scatter/gather capability. However, scatter/gather is available for all pointer dwords for all other descriptor types (provided the J bit is set).

## 10.3.4 Link Table Format

Link tables implement scatter/gather capability. For “gather” operations, a link table specifies a list of “memory segments” that are to be concatenated in the process of assembling parcels. For “scatter” operations, a link table specifies a list of memory segments into which the output data should be written. Scatter or gather of a parcel may be specified by a single link table or by a chain of link tables that are linked together with pointers, as shown in Figure 10-6.

A link table may contain any number of dword entries. Link table entry format is shown in Figure 10-5, and the field definitions are given in Table 10-9.

Access: Read/Write

| | 0–15 | 16–21 | 22 | 23 | 24–27 | 28–31 |
|---|---|---|---|---|---|---|
| R<br>W | SEGLEN | — | R | N | — | EPTR |
| Reset | All zeros | | | | | |

| | 32–63 |
|---|---|
| R<br>W | SEGPTR |
| Reset | All zeros |

**Figure 10-5. Link Table Entry**

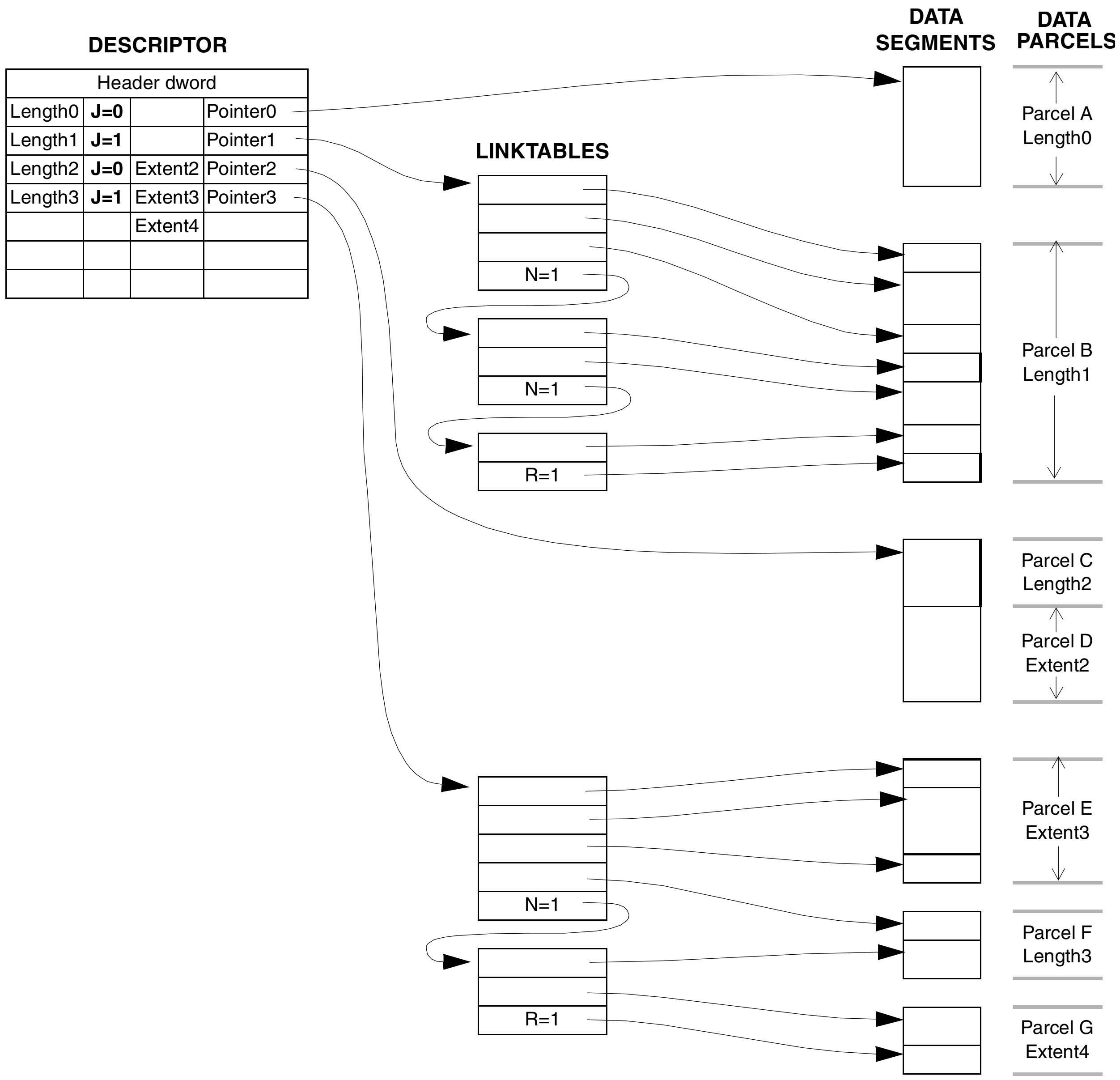
[1] Sometimes an EXTENT field refers to data in a pointer which is not in the same dword. For example, with the CCMP descriptor type (see Table 10-10), the length of the CRC check field appears in Extent0, but the field that is Extent0 bytes in length is referred to either by Pointer4 or Pointer5, depending on the direction bit in the descriptor’s header dword.

Table 10-9. Link Table Field Definitions

| Bits | Name | Description |
|---|---|---|
| 0-15 | SEGLEN | Length: 0-15<br>When N=0, SEGLEN is in the range 1 to 65535, specifying the number of bytes in the memory segment pointed to by SEGPTR. A value of 0 causes the SGZL error bit to be set in the Channel Status (see Section 10.4.4.2, "Channel Status Register (CSR)").<br>When N=1, SEGLEN must be 0. |
| 16-21 | — | Reserved |
| 22 | R | Return:<br>When N=0:<br>0 No special action.<br>1 Indicates the last entry in the chain of link tables. If this entry does not specify the right number of bytes to complete the last parcel, a G-STATE or S-STATE error is set in the Channel Status Register (see Section 10.4.4.2, "Channel Status Register (CSR)").<br>When N=1, ignored. |
| 23 | N | Next:<br>0 No special action.<br>1 Indicates the last dword in the current link table. The SEGPTR field is the address of the next link table in the chain. |
| 24-27 | — | Reserved |
| 28-31 | EPTR | Extended Pointer: Concatenated as the top 4 bits of the segment pointer when EAE is high (see the EAE bit in Table 10-11). |
| 32-63 | SEGPTR | Segment pointer: A memory address. |

There are two kinds of link table entries: "regular" entries or "next" entries (which have the N bit cleared or set, respectively). Each "regular" entry specifies a memory segment by means of a 36-bit starting address (SEGPTR) and a 16-bit length (SEGLEN). A "next" entry is used at the end of a link table to specify that the list of memory segments is continued in another link table. In a "next" entry, the N bit is set, the SEGPTR field gives the address of the next link table, and the SEGLEN field must be cleared. A chain of link tables may contain any number of link tables. Whether the list of memory segments is in a single link table or split into several link tables, the last entry in the last link table is a "regular" entry with the R (return) bit set. The R bit signifies the end of link table operations so that the channel returns to the descriptor for its next pointer (if any).

The construction and use of link tables is illustrated in Figure 10-6.



This figure illustrates various ways that a descriptor (table in upper left) may specify parcels:
The first pointer dword in the descriptor (following the header dword) specifies Parcel A using the simplest method—the parcel is specified directly through Pointer0 and Length0.
The next pointer dword uses a chain of link tables to specify Parcel B. Since J=1, Pointer1 is used as the address of a link table. The link table specifies several “regular” entries specifying data segments to be concatenated. The last word of the link table is a “next” entry indicating that the list continues in the next link table. The last entry in the last link table of the chain has the R bit set.
The last two cases illustrate how one pointer in a descriptor can be used to specify multiple parcels. Pointer2 and Length2 specify Parcel C, then Parcel D follows immediately afterwards, with length specified by Extent2. Pointer3 is used for three parcels (E, F, and G), this time using link tables.

**Figure 10-6. Descriptors, Link Tables, and Parcels**

As shown in Figure 10-6, in some cases a single parcel is accessed through a given POINTER, and the chain of link tables specifies only that parcel (this is the most common situation). In other cases, the descriptor POINTER is used multiple times to access a sequence of parcels, and the chain of link tables must supply data for the entire sequence.

### 10.3.4.1 Example of Link Table Operation

To further clarify the link table's operation, we explain in detail the case where the fourth pointer dword in the descriptor in Figure 10-6 is used to access parcels. We suppose that the descriptor type is such that Pointer3 is used to access successive parcels of size Extent3, Length3, and Extent4 respectively (refer to Table 10-10 for the significance of POINTER, EXTENT, and LENGTH fields in various descriptor types).

Since the J3 bit is set, Pointer3 is used as the address of a link table and not a data address. The channel begins by reading the first four dwords of the link table starting at Pointer3 into an internal "gather table buffer".

Using the first entry of the gather table buffer, the channel starts accessing the parcel by reading SEGLEN bytes beginning at SEGPTR. If the required parcel size (specified by 'Extent3' in the pointer dword) is greater than this first segment length, the channel moves on to the next entry of the gather table buffer, and reads SEGLEN bytes starting at SEGPTR. This process continues as long as there are more bytes to be read in the parcel. If all the link table entries in the channel's gather table buffer have been exhausted, then the channel reads the next four dwords of the link table into its gather table buffer. If a gather table buffer entry is encountered in which the N bit is set, the channel uses the SEGPTR field in that word to find the next link table in the chain.

Now assume that the channel accesses its next parcel using Pointer3 again, this time with length given by Length3. In this case the channel continues to the next line of the link table, and begins reading the memory segment specified there. As before, the channel concatenates memory segments from as many link table entries as necessary to obtain the required number of bytes (Length3).

Similarly, the next parcel is obtained by using Pointer3 yet again, this time with length given by Extent4.

Assume that for the current descriptor type, the Extent4 parcel is the last one to be accessed through Pointer3. Then the link table entry that supplies the last memory segment for Extent4 has the R bit set, signifying that this is the last entry in the chain of link tables.

**NOTE**

> The link table or chain of link tables accessed through a descriptor pointer must specify enough memory segments to hold precisely *all the data that will be accessed through that pointer*. This means that the combined lengths of the parcels associated with that pointer (where each parcel length is specified by a particular LENGTH or EXTENT field in the descriptor) must equal the combined lengths of the link table memory segments (SEGLEN fields). Otherwise the channel sets the error state in the SGLM bit of the channel status register (see Section 10.4.4.2, "Channel Status Register (CSR)").

### 10.3.5 Descriptor Types

Table 10-10 shows in summary form how the pointer dwords are used with the various descriptor types. Detailed information about each descriptor type is given in the remainder of this chapter. Additional explanation of the use of certain descriptor types, can be found in the SEC 3.0 Descriptor Programmer's Guide.

As in Table 10-7 above, older descriptor types which end in 0 are listed first, followed by newer types which end in 1.

**Table 10-10. Descriptor Format Summary**

| Descriptor Type | field type | Pointer Dword0 | Pointer Dword1 | Pointer Dword2 | Pointer Dword3 | Pointer Dword4 | Pointer Dword5 | Pointer Dword6 |
|---|---|---|---|---|---|---|---|---|
| 0000_0<br>aesu_ctr_<br>nosnoop | Length | reserved | Cipher Context In | Cipher Key | Main Data In | Data Out | Cipher Context Out | reserved |
| | Extent | reserved | reserved | reserved | reserved | reserved | reserved | reserved |
| 0001_0<br>common_<br>nosnoop for DES, KEU f8, RNGU, AES-CCM | Length | reserved | Context In | Key | Main Data In | Data Out | Context Out (incl. ICV out) | reserved |
| | Extent | reserved | reserved | reserved | reserved | reserved | reserved | reserved |
| 0001_0<br>common_<br>nosnoop for MDEU | Length | reserved | Context In | Key | Main Data In | ICV In | Context Out | reserved |
| | Extent | reserved | reserved | reserved | reserved | reserved | reserved | reserved |
| 0001_0<br>common_<br>nosnoop for AES-XCBC, AES-CMAC | Length | reserved | Context In | Key | Main Data In | reserved | Context Out | ICV Out |
| | Extent | reserved | reserved | reserved | reserved | ICV In | reserved | reserved |
| 0001_0<br>common_<br>nosnoop for KEU f9 | Length | reserved | Context In | Key | Main Data In | reserved | Context Out | ICV Out |
| | Extent | reserved | reserved | reserved | reserved | reserved | reserved | reserved |
| 0001_0<br>common_<br>nonsnoop for CRCU | Length | reserved | Context In | Key | Main Data In | reserved | Context Out | reserved |
| | Extent | reserved | reserved | reserved | reserved | ICV In | reserved | reserved |
| 0010_0<br>hmac_snoop<br>_no_afeu | Length | Hash Key | Hash-only Header | Cipher Key | Cipher Context In | Main Data In | Data Out | ICV Out |
| | Extent | reserved | reserved | reserved | reserved | reserved | reserved | reserved |
| 0101_0<br>common_<br>nosnoop_<br>afeu | Length | reserved | Context In (via In FIFO) | Cipher Key | Main Data In | Data Out | Context Out (via Out FIFO) | reserved |
| | Extent | reserved | reserved | reserved | reserved | reserved | reserved | reserved |

**Table 10-10. Descriptor Format Summary (continued)**

| Descriptor Type | field type | Pointer Dword0 | Pointer Dword1 | Pointer Dword2 | Pointer Dword3 | Pointer Dword4 | Pointer Dword5 | Pointer Dword6 |
|---|---|---|---|---|---|---|---|---|
| 1000_0<br>pkeu_mm | Length | “N” In | “B” In | “A” In | “E” In | “B” Out | reserved | reserved |
| | Extent | reserved | reserved | reserved | reserved | reserved | reserved | reserved |
| 1100_0<br>hmac_snoop_aesu_ctr | Length | Hash Key | Hash-only Header | AES Key | AES Context In | Main Data In | Data Out | ICV Out |
| | Extent | reserved | reserved | reserved | reserved | reserved | reserved | reserved |
| 0000_1<br>ipsec_esp | Length | HMAC Key | Hash-only Header | Cipher IV In | Cipher Key | Main Data In | Data Out | Cipher IV Out |
| | Extent | reserved | reserved | reserved | reserved | ICV In | ICV Out | reserved |
| 0001_1<br>802.11i AES ccmp | Length | CRC-only Header | AES Context In | AES Key | Hash-only Header | Main Data In | Data Out | AES Context Out |
| | Extent | CRC In/Out (FCS) | reserved | reserved | reserved | MIC In | MIC Out | reserved |
| 0010_1<br>srtp with ICV Check | Length | HMAC Key | AES Context In | AES Key | Main Data In | Data Out | HMAC Out | AES Context Out |
| | Extent | reserved | reserved | reserved | Hash-only Header | Hash-only Trailer | reserved | reserved |
| 0010_1<br>srtp without ICV Check | Length | HMAC Key | AES Context In | AES Key | Main Data In | HMAC In | Data Out | AES Context Out |
| | Extent | reserved | reserved | reserved | Hash-only Header | Hash-only Trailer | HMAC Out | reserved |
| 0011_1<br>pkeu_build | Length | “A0” In | “A1” In | “A2” In | “A3” In | “B0” In | “B1” In | "Build" Out |
| | Extent | reserved | reserved | reserved | reserved | reserved | reserved | reserved |
| 0100_1<br>pkeu_ptmul | Length | “N” In | “E” In | “Build” In | “B1” Out | “B2” Out | “B3” Out | reserved |
| | Extent | reserved | reserved | reserved | reserved | reserved | reserved | reserved |
| 0101_1<br>pkeu_ptadd_dbl | Length | “N” In | “Build” In | “B2” In | “B3” In | “B1” Out | “B2” Out | “B3” Out |
| | Extent | reserved | reserved | reserved | reserved | reserved | reserved | reserved |
| 1000_1 outbound<br>tls_ssl_block | Length | MAC Key | Cipher IV In | Cipher Key | Main Data In | Cipher-only Trailer | Data Out | Cipher IV Out |
| | Extent | reserved | reserved | reserved | Hash-only Header | ICV Out | reserved | reserved |
| 1000_1 inbound<br>tls_ssl_block | Length | MAC Key | Cipher IV In | Cipher Key | reserved | Main Data In | Data Out | Cipher IV Out |
| | Extent | reserved | reserved | reserved | Hash-only Header | ICV In | ICV Out | reserved |

**Table 10-10. Descriptor Format Summary (continued)**

| Descriptor Type | field type | Pointer Dword0 | Pointer Dword1 | Pointer Dword2 | Pointer Dword3 | Pointer Dword4 | Pointer Dword5 | Pointer Dword6 |
|---|---|---|---|---|---|---|---|---|
| 1001_1 outbound<br>tls_ssl_ stream | Length | MAC Key | Cipher IV In | Cipher Key | Main Data In | reserved | Data Out | Cipher IV Out |
| | Extent | reserved | reserved | reserved | Hash-only Header | ICV Out | reserved | reserved |
| 1001_1 inbound<br>tls_ssl_ stream | Length | MAC Key | Cipher IV In | Cipher Key | reserved | Main Data In | Data Out | Cipher IV Out |
| | Extent | reserved | reserved | reserved | Hash-only Header | ICV In | ICV Out | reserved |
| 1010_1<br>raid_xor | Length | Source F Data In | Source E Data In | Source D Data In | Source C Data In | Source B Data In | Source A Data In | Data Out |
| | Extent | reserved | reserved | reserved | reserved | reserved | reserved | reserved |
| 1011_1<br>ipsec_aes_ gcm | Length | AES Context In | AAD In | Nonce Part 2 In | AES Key In | Main Data In | Data Out | Cipher Context Out |
| | Extent | reserved | reserved | reserved | Nonce Part 1 In | AES ICV In | AES ICV Out | CRC ICV In/Out |
| 1100_1<br>dbl_crc | Length | Header In | Payload In | reserved | reserved | reserved | reserved | reserved |
| | Extent | Header ICV | Payload ICV | Header ICV Out | Payload ICV Out | reserved | reserved | reserved |
| others | | reserved | | | | | | |

## 10.4 Polychannel

The polychannel is the main control unit in the SEC. It implements four independent channels.

Each cryptographic task performed by the SEC is managed by a channel and makes use of one or more of the SEC's execution units (EUs). Control information and data pointers for a given task are stored in the form of a descriptor (see Section 10.3.1, "Descriptor Structure") placed in system memory. A descriptor determines what EUs are used, how they are configured, where to fetch needed data, and where to store the results.

The following subsections describe the operation (including descriptor processing, arbitration, and host notification), registers, and interrupts of the polychannel.

### 10.4.1 Channel Operation

#### 10.4.1.1 Channel Descriptor Processing

To invoke a cryptographic task, the host constructs a descriptor, selects a channel, and writes a pointer to the descriptor into the selected channel's fetch FIFO. Each fetch FIFO can store up to 24 pointers.

Typical operations performed by a channel to process a descriptor are:

1. Analyze the descriptor header to determine the cryptographic services required, and arbitrate for the appropriate EUs. If required EUs are already reserved by another channel, wait for the EUs to be available. When available, reserve them.
2. Set the mode register in each reserved EU(s) for the required EU function.
3. Fetch "parcels" (up to 64K–1 bytes long) from system memory using pointers from the descriptor buffer, and place them in either an EU input FIFO or EU registers, as appropriate. The term "parcel" refers here to any input or output of an EU algorithm, such as a key, hash result, input context, output context, or text data. "Context" refers to either an IV (initialization vector) or other internal EU state that can be read out or loaded in. "Text data" refers to plaintext or ciphertext to be operated on. Each parcel transfer may involve using link tables to gather input data that has been split into multiple segments in system memory.
4. Take data accumulated in the EU output FIFO and write it to system memory using pointers from the descriptor buffer. This may again involve using link tables to scatter output data into multiple segments in system memory.
5. If the data size is greater than EU FIFO size, continue fetching input data and writing output data to memory as needed.
6. After writing the last input data to each EU's input FIFO, write to the end of message register in the EU.
7. Wait for EU(s) to complete processing of text data.
8. Unload final results from output FIFOs and context registers and write them to external memory using pointers from the descriptor buffer. This may again involve using link tables to scatter output data into multiple segments in system memory.
9. Reset and release the EUs.
10. If enabled, then notify the host of descriptor completion (see Section 10.4.1.3, "Channel Host Notification").

### 10.4.1.2 Channel Arbitration

All channels share a set of common resources, including the EUs and the SEC's bus master interface (managed by the SEC controller). When multiple channels are used in parallel, arbitration may be required to determine which channel is serviced. The different arbitration schemes are described in Section 10.5.2, "Arbitration Algorithms."

Generally speaking, no arbitration for use of the controller/bus master interface is required. The channels within the polychannel execute one at a time, so individual channels do not experience contention when requests to the controller. In effect, when a channel wins arbitration for use of the polychannel, it wins use of the controller as well.

The same is not true of EUs. Once the controller has assigned an EU to a channel, that channel owns the EU for the duration of descriptor processing. The maximum amount of data that can be processed by a single descriptor is 64 Kbytes, which prevents a channel from owning an EU for an unbounded length of time.

While one channel owns a particular EU, it is possible for two or more other channels to request access to the same EU; in this case, an arbitration scheme determines which channel is granted next access. EU arbitration schemes are similar to channel arbitration mentioned above, and are described in Section 10.5.2, “Arbitration Algorithms.”

If a channel needs two EUs, a primary and a secondary, it requests them one at a time. Sometimes a channel reserves one EU and then has to wait for some other channel(s) to finish before obtaining the second requested EU. Though such waiting may occur, the requests are always eventually satisfied. Deadlock is avoided through the following design rules:

1. The channel always requests the secondary EU first.
2. In cases where both a primary and secondary are used, the choices for primaries and secondaries are distinct sets. Primaries are AESU, AFEU, DES, and KFEU, and the secondaries are MDEU and CRCU.

#### 10.4.1.3 Channel Host Notification

When a channel completes operation on a descriptor, it can notify the host that it is done through interrupt and/or through a writeback of the descriptor header dword. In case the descriptor operation is not completed or completed with a known error, the host may be notified by an error interrupt. The error interrupts, done interrupts and header writeback are described as follows:

- Error interrupts are always enabled at the channel level, but can be masked at the controller level. For more details concerning these interrupts, see Section 10.4.2.2, “Channel Error Interrupt.”
- Done interrupts are enabled on a per-channel basis by programming the channel’s configuration register. For programming details, refer to Section 10.4.2.1, “Channel Done Interrupt,” and Section 10.4.4.1, “Channel Configuration Register (CCR).”
- Independently of the done interrupt, channels can inform software of their completion status via header writebacks. Like done interrupts, writebacks are enabled on a per-channel basis by programming the CCR. If enabled, then upon completion the channel writes 0xFF to the DONE byte in the original descriptor header (see Table 10-5), allowing software to poll for completion of a specific descriptor. The CCR can also be programmed so that the channel writes back a status code indicating whether an integrity checking EU has encountered a mismatch between the received ICV and the recalculated ICV. Table 10-5 shows the specific bytes in the descriptor header that are updated in this case.

  For more details on programming the CCR for writeback, see Section 10.4.4.1, “Channel Configuration Register (CCR).”

**NOTE**

> The done and status writebacks are not performed should the channel signal any error during processing. For example, there are no writebacks in case of a failing, unmasked ICV check in an EU.

### 10.4.2 Channel Interrupts

Active channels can assert done and error interrupts to the controller. As with all SEC interrupt events, channel done and error interrupts are reflected in the controller’s interrupt status register. Channel do not have internal interrupt masks, but the controller can be programmed to disable channel interrupts through its interrupt enable register. For more details on interrupt types and disablement, see Section 10.5.4.2, “Interrupt Enable, Interrupt Status, and Interrupt Clear Registers (IER, ISR, ICR).”

#### 10.4.2.1 Channel Done Interrupt

Channel done interrupt generation depend on the setting of the CDIE (channel done interrupt enable) and NT (notification type) bits in the channel configuration register (see Section 10.4.4.1, “Channel Configuration Register (CCR)”). If both CDIE and NT are set, the channel generates an interrupt event after every successfully completed descriptor; If CDIE is set and NT is cleared, an interrupt is generated after each successfully completed descriptor with the DN (done notification) bit set in the descriptor’s header word. If the EU(s) signal any error during processing, the channel done interrupt is not generated.

Even if multiple channel done interrupt events are generated by a channel before the first can be cleared by the host, the interrupt events are not lost. The controller keeps count of the backlog of channel done interrupts from each channel (see Section 10.5.3, “Controller Interrupts”).

#### 10.4.2.2 Channel Error Interrupt

The channel error interrupt is generated when an error condition occurs during descriptor processing. The error could be a bus error for a transaction requested by the channel; or it could be in one of the EUs reserved by the channel, or in the channel itself. The channel error interrupt is asserted as soon as the error condition is detected. The type of error condition is reflected in the ERROR field of the channel status register (CSR).

For most error types, the error causes the corresponding channel to halt. Any EUs reserved by the halted channel continue to be reserved until the channel reset occurs. Other channels continue normal processing, though they may be held up if they need an EU that is reserved by a halted channel.

Handling of errors depends on the error type. Details of each error type are given in Table 10-15. For some types, the host must clear the source of the error before restarting the channel. If the channel is halted, the host restarts it by setting the no-pop-reset, continue or reset bits of the CCR (see Section 10.4.4.1, “Channel Configuration Register (CCR)”).

### 10.4.3 Polychannel Registers

The polychannel has several aggregate performance counters, which are common to all channels; plus a set of channel-specific registers, descriptor buffers, and link tables which are duplicated for each channel. The following subsections describes the format and function of all of these objects in the SEC’s memory.

#### 10.4.3.1 Traffic Counters

The SEC maintains several counters, which are described in the following subsections.

##### 10.4.3.1.1 Fetch FIFO Enqueue Counter

The fetch FIFO enqueue counter, shown in Figure 10-7, counts the total number of descriptor addresses that have been enqueued to the channel fetch FIFOs.

If the FETCH_FIFO_ENQ_COUNT field is 0x1111_1111, then adding another entry to the FIFO clears the register and causes the FFE_CNT bit (if enabled) to be set in the controller’s interrupt status register (see Section 10.5.4.2, “Interrupt Enable, Interrupt Status, and Interrupt Clear Registers (IER, ISR, ICR)”).

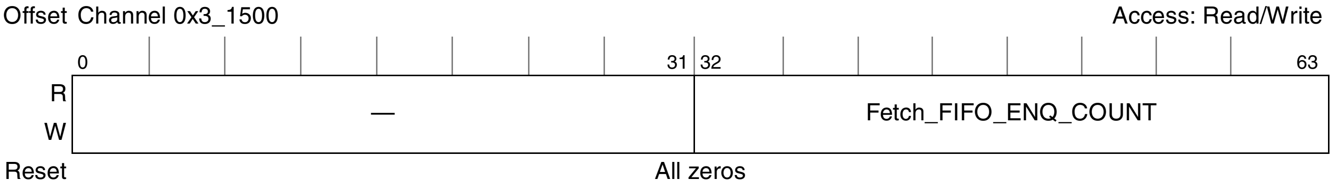


**Figure 10-7. Fetch FIFO Enqueue Counter**

### 10.4.3.1.2 Descriptor Finished Counter

The descriptor finished counter, shown in Figure 10-8, indicates the total number of descriptors that have successfully completed processing. It does not count descriptors that halt due to error.

When the DESCRIPTOR_FINISHED_COUNT field reaches 0x1111_1111, then the next completed descriptor clears the counter and causes the DF_CNT bit (if enabled) to be set in the controller’s interrupt status register (see Section 10.5.4.2, “Interrupt Enable, Interrupt Status, and Interrupt Clear Registers (IER, ISR, ICR)”).

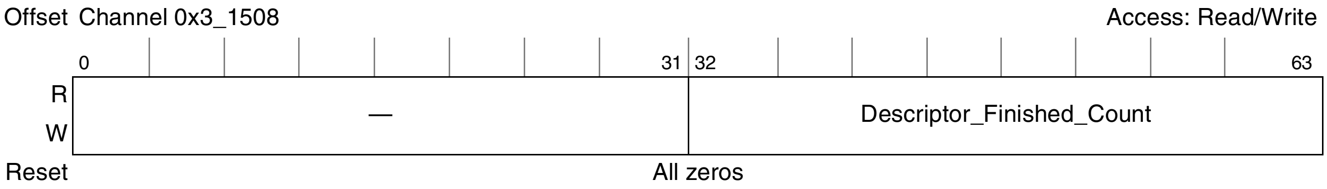


**Figure 10-8. Descriptor Finished Counter**

### 10.4.3.1.3 Data Bytes In Counter

The data bytes in counter, shown in Figure 10-9, indicates the total number of bytes written into a primary EU input FIFO. If other parcels such as context or ICV are placed in the input FIFO, they are not counted. When a secondary EU is used, data going only to the secondary EU (such as a hash-only region or authentication data) is counted, but the data used by both EUs is not double counted.

If this counter reaches all 1s, at the next count it rolls over to all 0s and the interrupt enable register’s DI_CNT bit is set (see Section 10.5.4.2, “Interrupt Enable, Interrupt Status, and Interrupt Clear Registers (IER, ISR, ICR)”).

If this counter is read by software in 32-bit increments, then the least significant 32 bits must be read first, followed by the most significant 32 bits. If this counter is written by software in 32 bit increments, then the most significant 32 bits must be written first, followed by the least significant 32 bits. Note that 32 bit reads and writes must not be interleaved (that is, read low, write low, read high, write high is not allowed). These restrictions are required to maintain counter coherency.

Offset Channel 0x3_1510 Access: Read/Write

| | 0–31 | 32–63 |
|---|---|---|
| R<br>W | Data_Bytes_In_Count | |
| Reset | All zeros | |

**Figure 10-9. Data Bytes In Counter**

### 10.4.3.1.4 Data Bytes Out Counter

The data bytes out counter, shown in Figure 10-10, indicates the total number of payload bytes read from an EU output FIFO. If other parcels such as context or ICV are read from the output FIFO, they are not counted. In no case is data counted twice by the same counter.

If this counter reaches all 1s, at the next count it rolls over to all 0s and the interrupt enable register's DO_CNT bit is set (see Section 10.5.4.2, "Interrupt Enable, Interrupt Status, and Interrupt Clear Registers (IER, ISR, ICR)").

If this counter is read by software in 32-bit increments, then the least significant 32 bits must be read first, followed by the most significant 32 bits. If this counter is written by software in 32 bit increments, then the most significant 32 bits must be written first, followed by the least significant 32 bits. Note that 32 bit reads and writes must not be interleaved (that is, read low, write low, read high, write high is not allowed). These restrictions are required to maintain counter coherency.

Offset Channel 0x3_1518 Access: Read/Write

| | 0–31 | 32–63 |
|---|---|---|
| R<br>W | Data_Bytes_Out_Count | |
| Reset | All zeros | |

**Figure 10-10. Data Bytes Out Counter**

## 10.4.4 Channel Registers

The channel registers are replicated for each of the 4 channels in the polychannel.

### 10.4.4.1 Channel Configuration Register (CCR)

This register contains bits that allow the user to configure and reset the channel. The CCR fields are shown in Figure 10-11, and described in Table 10-11.

Offset Channel 1: 0x3_110C
Channel 2: 0x3_120C
Channel 3: 0x3_130C
Channel 4: 0x3_140C
Channel 1: 0x3_110C
Channel 2: 0x3_120
Channel 3: 0x3_130C
Channel 4: 0x3_140C

Access: Read/Write

| | 0–28 | 29 | 30 | 31 |
|---|---|---|---|---|
| R<br>W | — | NPR | CON | R |
| Reset | All zeros | | | |

| | 32–50 | 51 | 52–54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | PBS | — | BS | IWSE | — | EAE | CDWE | AWSE | NT | CDIE | — |
| Reset | All zeros | | | | | | | | | | | |

**Figure 10-11. Channel Configuration Register (CCR)**

**Table 10-11. Channel Configuration Register Fields**

| Bits | Name | Description |
|---|---|---|
| 0–28 | — | Reserved, should be set to zero. |
| 29 | NPR | No-Pop-Reset[1].<br>0 No special action.<br>1 Causes the same channel reset actions as the CON bit, except that the fetch FIFO is left unchanged, such that the channel picks up by re-fetching the previous descriptor. This permits debug of a descriptor in-place without having to rewrite the descriptor pointer into the fetch FIFO.<br>• If the NPR bit is set while the channel is requesting an EU assignment from the controller, the channel cancels its request.<br>• If the NPR bit is set after the channel has been assigned one or more EUs, the channel requests a write from the controller to set the software reset bit of each reserved EU. The channel then releases the EU(s). |
| 30 | CON | Continue bit[1].<br>0 No special action.<br>1 Causes the same channel reset actions as bit R, except that the fetch FIFO and bits 32-63 of the CCR register are not cleared. After the reset sequence is complete, this bit automatically returns to 0 and the channel resumes normal operation, servicing the next descriptor pointer in the fetch FIFO, if any.<br>• If the CON bit is set while the channel is requesting an EU assignment from the controller, the channel cancels its request.<br>• If the CON bit is set after the channel has been assigned one or more EUs, the channel requests a write from the controller to set the software reset bit of each reserved EU. The channel then releases the EU(s). |

Table 10-11. Channel Configuration Register Fields (continued)

| Bits | Name | Description |
|---|---|---|
| 31 | R | Reset channel[1].<br>0 No special action.<br>1 Causes a software reset of the channel. All channel registers are cleared. Other actions depend on the state of the channel when the bit is set:<br>• If the R bit is set while the channel is requesting an EU assignment from the controller, then the channel cancels its request.<br>• If the R bit is set after the channel has been assigned one or more EUs, the channel requests a write from the controller to set the software reset bit of each reserved EU. The channel then releases the EU(s).<br>After the reset sequence is complete, the channel returns to the idle state, the R bit is cleared automatically, and normal operation is resumed. |
| 32–48 | — | Reserved, should be set to zero. |
| 49 | FCC | Fast clock counting.<br>0 Watchdog timer counts normally<br>1 Watchdog timer counts in an accelerated fashion (force-assert several selected bits in timer) to assist with functional testing. |
| 50 | WGN | Watchdog go now.<br>0 Watchdog timer disabled<br>1 Watchdog timer enabled |
| 51 | PBS | Permit byte summing.<br>0 Bytes written to EU input FIFOs and read from EU output FIFOs are not counted in the data bytes counters<br>1 Bytes written to EU input FIFOs and read from EU output FIFOs are counted in the data bytes counters |
| 52–54 | — | Reserved, should be set to zero. |
| 55 | BS | Burst size. The SEC accesses long text-parcels in main memory through bursts of programmable size.<br>0 Burst size is 64 bytes<br>1 Burst size is 128 bytes |
| 56 | IWSE | ICV writeback status enable.<br>0 No special action.<br>1 If the descriptor calls for ICV checking, then at the completion of descriptor processing, the channel writes back to the descriptor header the DONE, ICCR0, and ICCR1 fields (see Table 10-5).[2] |
| 57 | — | Reserved, should be set to zero. |
| 58 | EAE | Extend address enable. This bit determines whether the channel uses a 36-bit address bus or a 32-bit address bus.<br>0 Channel's address bus is 32 bits.<br>1 Channel's address bus is 36 bits. |
| 59 | CDWE | Channel done writeback enable.<br>0 Channel done writeback disabled.<br>1 Channel done writeback enabled. Upon successful completion of descriptor processing, if the NT bit is cleared (for global notification), or if the DN (done notification) bit is set in the header word of the descriptor, then the channel notifies the host by writing back the descriptor header with the DONE field shown in Table 10-5. This enables the host to poll the memory location of the original descriptor header to determine if that descriptor has been completed.[2] |

Table 10-11. Channel Configuration Register Fields (continued)

| Bits | Name | Description |
|---|---|---|
| 60 | AWSE | Always writeback status enable.<br>0 No special action.<br>1 At the completion of processing each descriptor, the channel writes back to the descriptor header the DONE, ICCR0, and ICCR1 fields (see Table 10-5). In this case, IWSE has no effect.[2] |
| 61 | NT | Notification type. This bit controls when the channel generates channel done notification. Channel done notification can take the form of an interrupt and/or modified header writeback, depending on the state of the CDIE and CDWE control bits.<br>0 Global notification: The channel generates channel done notification (if enabled) at the end of each descriptor.<br>1 Selected notification: The channel generates channel done notification (if enabled) at the end of every descriptor with the DN bit set in the descriptor header. |
| 62 | CDIE | Channel done interrupt enable.<br>0 Channel done interrupt disabled<br>1 Channel done interrupt enabled. Upon successful completion of descriptor processing, if the NT bit is cleared (for global notification), or if the DN (done notification) bit is set in the header word of the descriptor, then a channel done interrupt is asserted to notify the host.[2]<br>Refer to Section 10.4.4, "Channel Registers," for a complete description of channel done interrupt operation. |
| 63 | — | Reserved, should be set to zero. |

[1] WARNING: When using reset bits R, CON and NPR: the configuration register must be polled to confirm completion of the multi-cycle reset sequence. The length of time required for this reset sequence depends on several factors and should be considered indeterminate. Completion is indicated by the self-clearing of the asserted reset bit. Failure to ensure completion of reset prior to writing to the channel may result in a channel hang condition.

[2] WARNING: The done interrupt, done writeback, and status writeback do not occur if an EU produces an error interrupt to the channel. In particular, if the ICV check error interrupt is enabled in the EU (see the ICE bit in the EU's interrupt mask register), and the ICV check finds a mismatch, then the channel produces an error interrupt but no channel done interrupt or writebacks.

Table 10-12 shows the CCR and descriptor header bit settings for different descriptor header writeback options; and Table 10-13 shows the bit settings for different done interrupt generation options.

Table 10-12. Writeback Options

| AWSE<br>CCR bit 60 | CDWE<br>CCR bit 59 | IWSE<br>CCR bit 56 | NT<br>CCR bit 61 | DN<br>Header bit 63 | Writeback Action for a Descriptor completing without error |
|---|---|---|---|---|---|
| 1 | x | x | x | x | write back header fields DONE, ICCR0, ICCR1 |
| 0 | 1 | x | 1 | 0 | no writeback performed |
| 0 | 1 | x | 1 | 1 | write back header field DONE |
| 0 | 1 | x | 0 | x | write back header field DONE |
| 0 | x | 1 | x | x | if the descriptor header indicates ICV checking in AESU, CRCU, KEU, or MDEU, then write back header fields DONE, ICCR0, and ICCR1. |

Table 10-13. Done Interrupt Options

| NT<br>CCR bit 61 | DN<br>Header bit 63 | CDIE<br>CCR bit 62 | Done Interrupt action by channel to controller for a descriptor completing without error |
|---|---|---|---|
| x | x | 0 | never assert done interrupt |
| 0 | x | 1 | assert done interrupt |
| 1 | 0 | 1 | never assert done interrupt |
| 1 | 1 | 1 | assert done interrupt |

### 10.4.4.2 Channel Status Register (CSR)

CSR contains status fields and counters which provide status information regarding the channel’s processing of the current descriptor. This register is intended for debug use.

Figure 10-12 shows the channel status register fields, which are described in Table 10-14.

The multiple state-machine architecture of the channel makes it difficult to completely determine the channel’s status. The channel should be considered idle only if GET_STATE, PUT_STATE, and MAIN_STATE and FF_LEVEL are all cleared.

Offset Channel_1 0x01110
Channel_2 0x01210
Channel_3 0x01310
Channel_4 0x01410
Channel 1: 0x3_1114
Channel 2: 0x3_1214
Channel 3: 0x3_1314
Channel 4: 0x3_1414

Access: Read/Write

| | 0 | 1–7 | 8 | 9–15 | 16–22 | 23–31 |
|---|---|---|---|---|---|---|
| R<br>W | — | GET_STATE | — | PUT_STATE | — | MAIN_STATE |
| Reset | All zeros | | | | | |

| | 32–34 | 35–39 | 40–43 | 44 | 45 | 46 | 47 | 48–59 | 60–63 |
|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | FF_LEVEL | — | PRD | SRD | PD | SD | Error | — |
| Reset | All zeros | | | | | | | | |

Figure 10-12. Channel Status Register (CSR)

Table 10-14. Channel Status Register Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0 | — | Reserved. |
| 1–7 | GET_STATE | Get state machine state. This field reflects the state of the get state machine when it last went to sleep, or the state captured when an error occurred. For debug purposes only. |

**Table 10-14. Channel Status Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 8 | — | Reserved. |
| 9–15 | PUT_STATE | Put state machine state. This field reflects the state of the put state machine when it last went to sleep, or the state captured when an error occurred. For debug purposes only. |
| 16–22 | — | Reserved. |
| 23–31 | MAIN_STATE | Main state machine state. This field reflects the state of the main state machine when it last went to sleep, or the state captured when an error occurred. For debug purposes only. |
| 32–34 | — | Reserved, should be set to zero. |
| 35–39 | FF_LEVEL | Fetch FIFO level. This five-bit counter indicates how many pointers are currently stored in the fetch FIFO. |
| 40-43 | — | Reserved, should be set to zero. |
| 44 | PRD | Primary EU reset done. This bit reflects the state of the reset done signal from the assigned primary EU.<br>0 The assigned primary EU reset done signal is inactive.<br>1 The assigned primary EU reset done signal is active, indicating its reset sequence has completed and it is ready to accept data. |
| 45 | SRD | Secondary EU reset done. This bit reflects the state of the reset done signal from the assigned secondary EU.<br>0 The assigned secondary EU reset done signal is inactive.<br>1 The assigned secondary EU reset done signal is active, indicating its reset sequence has completed and it is ready to accept data. |
| 46 | PD | Primary EU done. This bit reflects the state of the done interrupt from the assigned primary EU.<br>0 The assigned primary EU done interrupt is inactive.<br>1 The assigned primary EU done interrupt is active, indicating the EU has completed processing and final values are available from EU registers. If the EU has an output FIFO, then all text data output has been placed in the output FIFO. If the EU provides context out through the output FIFO, then the context is placed in the output FIFO *after* the PD bit is asserted. |
| 47 | SD | Secondary EU done. The SEC_DONE bit reflects the state of the done interrupt from the assigned<br>secondary EU.<br>0 The assigned secondary EU done interrupt is inactive.<br>1 The assigned secondary EU done interrupt is active, indicating the EU has completed processing and final values are available from EU registers. |
| 48–59 | Error | Error bits for the channel. See Figure 10-15. |
| 60–63 | — | Reserved. |

Table 10-15 lists the errors corresponding to each bit in the CSR’s Error field. Multiple bits may be set simultaneously. Whenever an error field bit is set a channel error interrupt is generated, and in most cases the channel is halted. For some error types, the host must take action to clear the error bit before restarting the channel, as described in Table 10-15. For information about restarting the channel, see the description of the R and CON bits in Section 10.4.4.1, “Channel Configuration Register (CCR)”.

**Table 10-15. Channel Status Register Error Field Definitions**

| CSR Bit # | Name | Error |
|---|---|---|
| 48 | DOF | Double Fetch FIFO write overflow error. This bit is set when the channel fetch FIFO is full, SOF is set, and another write has been made to the fetch FIFO. This error halts the channel. To clear this error, the host must write a '1' to this bit. |
| 49 | SOF | Single Fetch FIFO write overflow error. This bit is set when the channel fetch FIFO is full and another write has been made to the fetch FIFO. The channel continues processing, but the descriptor pointer is lost. To clear this error, the host must write a '1' to this bit. |
| 50 | MDTE | Master Data Transfer Error. When the SEC, while acting as a bus master, detects an error, the controller passes this error to the channel. This error halts the channel. Restarting the channel clears this bit. |
| 51-52 | | Reserved |
| 53 | IDH | Illegal descriptor header. Possible causes of an illegal descriptor header are:<br>• Invalid primary EU indicated by op0 field in descriptor header.<br>• Invalid secondary EU indicated by op1 field in descriptor header.<br>This error halts the channel. Restarting the channel clears this bit. |
| 54 | | Reserved |
| 55 | EUE | EU error. An EU assigned to this channel has generated an error interrupt. This error may also be reflected in the controller's interrupt status register. This error halts the channel. To clear this error, the host must clear the error source in the EU that produced the error. |
| 56 | WDT | Watchdog timeout. The main state machine stayed asleep too long. This timer runs only after EUs have been reserved, and does not run if the primary EU is the RNGU or PKEU. The timeout interval is controlled by the FCC field of the Channel Configuration Register. This error halts the channel. Restarting the channel clears this bit. |
| 57 | SGLM | Scatter/Gather Length Mismatch. Indicates the total data size covered by a gather link table did not match the total data size from the main descriptor. This error halts the channel. Restarting the channel clears this bit. |
| 58 | RSI | RAID Size Incorrect. The channel was provided with a descriptor of type RAID_XOR with data sizes not permitted. To clear this error, the host must write a '1' to this bit. |
| 59 | RSG | RAID Scatter Gather Error. The channel was provided with a descriptor of type RAID_XOR with a j bit set. Use of scatter/gather is not permitted with RAID_XOR type descriptors. To clear this error, the host must write a '1' to this bit. |

### 10.4.4.3 Current Descriptor Pointer Register (CDPR)

The CDPR reflects the value of the head end of the fetch FIFO, which contains the address of the descriptor which the channel is currently processing.

CPDR fields are shown in Figure 10-13, and described in Table 10-16.

Offset Channel 1: 0x3_1140, Channel 2: 0x3_1240,
Channel 3: 0x3_1340, Channel 4: 0x3_1440

Access: Read only

| | 0–15 | 16–27 | 28–31 | 32–63 |
|---|---|---|---|---|
| R | ID_TAG | — | EPTR | CUR_DES_PTR_ADRS |
| W | | | | |
| Reset | All zeros | | | |

**Figure 10-13. Current Descriptor Pointer Register**

Table 10-16. Current Descriptor Pointer Register Fields

| Bits | Name | Description |
|---|---|---|
| 0–27 | — | Reserved, must be cleared. |
| 28-31 | EPTR | Extended Pointer: Concatenated as the top 4 bits of the CUR_DES_PTR_ADRS when EAE is high (see the EAE bit description in Table 10-11). |
| 32–63 | CUR_DES_PTR_ADRS | Current Descriptor Pointer Address. Pointer to system memory location of the current descriptor. This field reflects the starting location in system memory of the descriptor currently loaded into the DB. This value is updated whenever the channel requests a fetch of a descriptor from the controller.<br>The value from the fetch FIFO is transferred to the current descriptor pointer register immediately after the fetch is completed.<br>This address is used as the destination for writeback of the modified header dword, if header writeback notification is enabled. |

## 10.4.4.4 Fetch FIFO Enqueue Register (FFER)

Each channel contains a fetch FIFO to store a queue of pointers to descriptors which the channel will process. A pointer is added to the queue by writing to the FFER.

The register is shown in Figure 10-14, and the fields are described in Table 10-17.

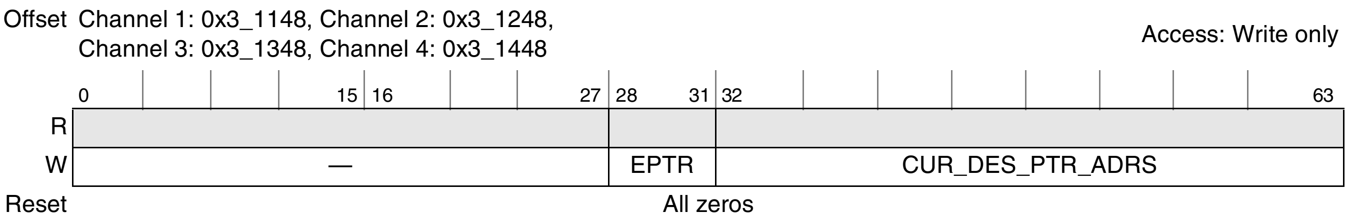

Offset Channel 1: 0x3_1148, Channel 2: 0x3_1248,
Channel 3: 0x3_1348, Channel 4: 0x3_1448
Access: Write only
0
15
16
27
28
31
32
63
R
W
—
EPTR
CUR_DES_PTR_ADRS
Reset
All zeros

Figure 10-14. Fetch FIFO Enqueue Register (FFER)

Table 10-17. Fetch FIFO Enqueue Register Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–27 | — | Reserved, must be cleared. |
| 28-31 | EPTR | Extended Pointer: Concatenated as the top 4 bits of the FETCH_ADR when EAE is high (see the EAE bit description in Table 10-11). |
| 32–63 | FETCH_ADR | Fetch Address. Pointer to system memory location of the first byte of descriptor to be processed. |

In channel-driven access, the host CPU creates a descriptor in memory containing all relevant mode and location information for the SEC, then launches the descriptor by writing its address to the fetch FIFO enqueue register.

The fetch FIFO can hold up to 24 descriptor pointers at a time. When the current descriptor’s processing is finished, the next fetch FIFO entry is read and the descriptor located at FETCH_ADR is launched.

NOTE

When extended addresses are enabled (by setting the EAE bit in channel configuration register), then the FFER's EPTR field must be written before or concurrently with the FETCH_ADR field. This is necessary because writing the least significant byte (bits 56–63) is the "trigger" which causes the FFER contents to be added to the FIFO.

## 10.4.5 Channel Buffers and Tables

Besides the registers described in Section 10.4.4, "Channel Registers," each channel has memory allocated for descriptors and scatter/gather link table entries (described in Section Section 10.3, "Descriptors"). The following subsections describe these features.

### 10.4.5.1 Descriptor Buffer (DB)

The descriptor buffer (DB) provides read-only access to the descriptor currently being processed by the channel. All descriptors are 8 dwords long. For descriptor format, see Figure 10-2. The address ranges of each channel's DB are shown in Table 10-3.

Note that the DB is working storage and the channel may modify the contents of the DB during processing. In debug scenarios, it may be useful to read the contents of the DB to determine if a well formed descriptor is being fetched by the channel. Potential causes of malformed descriptors in the DB include:

- The descriptor is built incorrectly
- The descriptor is fully or partially overwritten by some other system bus master before the SEC can fetch the descriptor
- The descriptor is not built at the address written to the fetch FIFO

### 10.4.5.2 Scatter and Gather Link Tables (SLT, GLT)

A pointer dword in the descriptor buffer (DB) refers to a Gather Link Table (GLT) or a Scatter Link Table (SLT) if the J bit in the dword is set. As a channel works on a DB pointer entry, the GLT/SLT is loaded into channel memory. Reads from the GLT/SLT are enabled for debug purposes.

Figure 10-15 summarizes the entry format and address ranges for gather and scatter link table entries.

Offset Channel 1: 0x3_11c0-0x3_11df (Gather); 0x3_11e0-0x3_11ff(Scatter)
Channel 2: 0x3_12c0-0x3_12df (Gather); 0x3_12e0-0x3_12ff (Scatter)
Channel 3: 0x3_13c0-0x3_13df (Gather); 0x3_13e0-0x3_13ff (Scatter)
Channel 4: 0x3_14c0-0x3_14df (Gather); 0x3_14e0-0x3_14ff (Scatter)

Access: Read/Write

| | 0–15 | 16–21 | 22 | 23 | 24–27 | 28–31 |
|---|---|---|---|---|---|---|
| R<br>W | SEGLEN | — | R | N | — | EPTR |
| Reset | Not reset | | | | | |

| | 32–63 |
|---|---|
| R<br>W | SEGPTR |
| Reset | Not reset |

**Figure 10-15. Gather/Scatter Link Table Entry Format and Memory Ranges**

## 10.5 Controller

All transfers between the hosts and the EUs are moderated by the controller. Some of the main functions of the controller are as follows:

- Accept and execute commands from the slave system bus to read or write memory-mapped locations (up to 64 bits) anywhere in the SEC.
- Accept and execute requests from the polychannel to transfer blocks of bytes among system memory, EUs, and the channels.
- Arbitrate between channels when they contend for EUs and bus access
- Realign read and write data to the proper byte alignment
- Monitor interrupts from channels and pass them to the host

The remainder of this section discusses the controller's bus management, arbitration, interrupts, and registers.

### 10.5.1 Bus Transfers

As shown in Figure 10-1, the SEC has an internal bus and connects to the SoC's system bus. The internal bus is a private 64-bit slave bus, with the controller block as the sole master. The SoC's system bus actually refers to two buses: a slave bus and a master bus, for which SEC's controller operates as slave and master, respectively. All accesses to SEC over the system bus go through the controller.

As mentioned in Section 10.1.3, "Controller Overview", there are two modes of access to the SEC, depending on whether the SEC's controller is slave or master. These two modes of access (host-controlled and channel-controlled) are discussed in the following subsections.

#### 10.5.1.1 Host-Controlled Access

For host-controlled access, the host uses the SoC's slave bus to access the controller as a slave, and the controller relays the read or write accesses over the internal bus to the appropriate registers and FIFOs of the EUs. When a write command is received from the system bus, the controller takes the data and sends it to whichever internal location is indicated by the address. For a read, the controller goes to the internal location, fetches the requested data from the specified address (if allowed), and returns it over the system bus.

Host-controlled access is much more CPU-intensive than channel-controlled access, and requires a great deal of familiarity with the EU and controller registers and procedures. If host-controlled access is used, it is recommended that only a single EU be operated at a time. Snooping is not available through this interface.

**NOTE**

> Host-controlled access of execution units is provided primarily for system debug purposes. The SEC contains no mechanism to arbitrate between host and channel accesses to EUs. Simultaneous use of an execution unit by a channel and a host is liable to force the execution unit into an error condition.

### 10.5.1.2 Channel-Controlled Access

Channel-controlled access is the SEC's normal operating mode. The controller performs data transfers based on information from the channels' descriptors. The controller can queue up to four requests. The controller dequeues requests and performs the required transfer. Most transfers involve not only the internal bus, but also the SoC's master  bus with the controller as bus master.

When the SEC performs a read or write transaction as master, in some cases the intended target (for instance, system memory) may terminate the transaction due to an error. Once the transaction is posted to the SoC's target queue, it is the SoC's responsibility to either complete the transaction or signal an error. An error in an SEC-initiated transaction is also reported by the SEC through the channel interrupt status register (ISR). The host is able to determine which channel generated the interrupt by checking the ISR for the channel ERROR bit.

#### 10.5.1.2.1 Channel Controlled Read—Detailed Description

A detailed description for a system bus read with controller as master is as follows:

1. Channel asserts bus read request to the controller
2. Channel furnishes external read address, internal write address, and transfer length
3. Controller asserts request to the system bus through the master interface
4. Controller waits for system bus read to begin
5. When bus read begins, controller receives data from the master interface and performs a write to the appropriate internal address supplied by the channel. Data may be realigned byte-wise by the controller if either:
   — the external read address was not on an 8-byte boundary, or
   — the internal write address was not on an 8-byte boundary.
6. Transfer continues until the bus read is completed and the controller has written all data to the appropriate internal address. The master interface continues making bus requests until the full data length has been read.

#### 10.5.1.2.2 System Bus Master Write—Detailed Description

A detailed description for a system bus write with controller as master is as follows:

1. Channel asserts its bus write request to the controller.
2. Channel furnishes internal read address, external write address, and transfer length.
3. Controller performs a read from the appropriate internal address supplied by the channel, loads the write data into its FIFO, asserts a request to the system bus through the master interface, and waits for the system bus to become available.
4. When the system bus becomes available, controller writes data from its FIFO to the master interface.

## 10.5.2 Arbitration Algorithms

This section applies to both arbitration for use of the polychannel, and arbitration for use of execution units. Control fields for both are in the master control register (Section 10.5.4.6, “Master Control Register (MCR)”), as follows:

- CHN3_BUS_PR_CNT and CHN4_BUS_PR_CNT control polychannel arbitration
- CHN3_EU_PR_CNT and CHN4_EU_PR_CNT control EU arbitration

In this section we refer to generic control fields CHN3_*xx*_PR_CNT and CHN4_*xx*_PR_CNT, where “*xx*” refers to either “BUS” or “EU”.

If both CHN3_*xx*_PR_CNT and CHN4_*xx*_PR_CNT are zero (the default), the arbitration is round-robin (see Section 10.5.2.1); otherwise a weighted priority scheme is used (see Section 10.5.2.2).

### 10.5.2.1 Round-Robin Arbitration

In round-robin arbitration, requesting channels are granted access in rotating numerical order: 1, 2, 3, 4, 1, 2, ... etc.

### 10.5.2.2 Weighted Priority Arbitration

In the weighted priority scheme, the priority is as follows:

- Channel 1—Highest priority
- Channel 2—Second highest priority, unless CHN3_*xx*_PR_CNT or CHN4_*xx*_PR_CNT has expired
- Channel 3—Third priority, unless CHN4_*xx*_PR_CNT expired
- Channel 4—Lowest priority, until CHN4_*xx*_PR_CNT expired

Initially, the priority is fixed from highest to lowest as channel 1, channel 2, channel 3, and channel 4, in that order. When channel 3 has lost arbitration the number of times specified in CHN3_*xx*_PR_CNT, channel 3 replaces channel 2 as the second-highest priority in the next round of arbitration. Likewise, when channel 4 has lost arbitration the number of times specified in CHN4_*xx*_PR_CNT, channel 4 replaces channel 2 as the second-highest priority in the next round of arbitration. These rules prevent channels 3 and 4 from being locked out.

Channel 1 always has the highest priority, but cannot make back-to-back requests. It follows that the second highest priority channel wins arbitration either immediately, or after one win for channel 1.

Note that the SEC does not dynamically adjust its own transaction priorities. System software, however, can adjust SEC transaction priority in real time, with the change in priority taking effect immediately.

## 10.5.3 Controller Interrupts

### 10.5.3.1 Controller Interrupt Conditions and Interrupt Generation

All interrupt outputs from other SEC blocks are fed to the controller as interrupt conditions. In addition, the controller itself detects some interrupt conditions. The controller maintains an interrupt status register (ISR) with bits corresponding to all of these possible interrupt conditions. If an interrupt condition occurs and the corresponding bit of the interrupt enable register (IER) is set, then the associated ISR bit is set, indicating the presence of a pending interrupt.

A channel can generate frequent interrupts, especially if it is configured to interrupt at the completion of each descriptor. To make sure that the host receives the right number of interrupts, each channel done interrupt has a special “queuing” feature. If multiple channel done interrupts are generated before the first is cleared, then the additional interrupts are counted by the controller. Each time the host clears a channel interrupt, the count is decremented. If the host clears the channel interrupt and the count reaches zero, the channel done interrupt is negated. If the count does not reach zero, the controller negates the interrupt for one cycle and then re-asserts it.

Up to 15 interrupts can be queued for each channel. If the count of queued interrupts for any channel exceeds 15, then that channel’s done overflow bit is set in the channel’s ISR (if the corresponding IER bit is set), and the channel done interrupt is asserted.

### 10.5.3.2 Blocking of Interrupts

Interrupt conditions from the channels and controller can only be blocked through the controller’s IER, as described in Section 10.5.3.1. However, the EU interrupt conditions may be blocked at two different levels. There is an interrupt mask register in each EU which can block particular interrupt conditions before they reach the EU’s interrupt status register. In addition, interrupts from EUs can be individually blocked by bits of the controller’s IER before they reach the controller’s ISR. For normal operation, interrupts from EUs are typically disabled in the controller’s IER, but they still reach the channel, and the channel produces done or error interrupts to the host as needed.

### 10.5.3.3 Interrupt Handling

To handle an interrupt, the host must read the ISR to determine the source. If necessary, the host may read the interrupt status registers of other blocks to ascertain the cause. In some cases, the host may need to take action to clear the root cause of the interrupt. Once the appropriate action is taken, the host can clear the ISR bit by setting the corresponding bit of the interrupt clear register (ICR). If the cause of the interrupt condition has not been cleared, or if there is another interrupt condition from the same source, then the ISR bit clears for a cycle and then goes high again, and the interrupt signal to the host remains high. If the ISR bit is successfully cleared and no other interrupt conditions are present, the controller negates its interrupt signal. If any interrupts are still pending in the ISR, the interrupt remains asserted.

## 10.5.4 Controller Registers

The controller registers are described in detail in the following sections.

### 10.5.4.1 EU Assignment Status Register (EUASR)

The EUASR indicates which EUs are reserved by a particular channel. When an EU is already assigned, it is inaccessible to any other channel.

The EAUSR fields are displayed in Figure 10-16. The register has a four-bit field for each EU which indicates the EU's assigned channel. The field values and corresponding channel assignments are shown in Table 10-18.

Offset 0x3_1028 Access: Read only

| | 0 3 | 4 7 | 8 11 | 12 15 | 16 19 | 20 23 | 24 27 | 28 31 |
|---|---|---|---|---|---|---|---|---|
| R | — | AFEU | — | MDEU | — | AESU | — | DEU |
| W | | | | | | | | |
| Reset | 1 1 1 1 | 0 0 0 0 | 1 1 1 1 | 0 0 0 0 | 1 1 1 1 | 0 0 0 0 | 1 1 1 1 | 0 0 0 0 |

| | 32 | | | 47 | 48 51 | 52 55 | 56 59 | 60 63 |
|---|---|---|---|---|---|---|---|---|
| R | — | — | CRCU | — | — | PKEU | — | RNG |
| W | | | | | | | | |
| Reset | 1 1 1 1 | 1 1 1 1 | 0 0 0 0 | 0 0 0 0 | 1 1 1 1 | 0 0 0 0 | 1 1 1 1 | 0 0 0 0 |

**Figure 10-16. EU Assignment Status Register (EUASR)**

**Table 10-18. Channel Assignment Value**

| Value | Channel |
|---|---|
| 0x0 | No channel assigned |
| 0x1 | Channel 1 |
| 0x2 | Channel 2 |
| 0x3 | Channel 3 |
| 0x4 | Channel 4 |
| 0xA–0xE | Undefined |
| 0xF | Unavailable |

### 10.5.4.2 Interrupt Enable, Interrupt Status, and Interrupt Clear Registers (IER, ISR, ICR)

The SEC controller generates the interrupt outputs from all possible interrupt sources. These outputs are enabled, displayed, and cleared by the IER, ISR, and ICR, respectively. These three registers share a common set of bit fields, which are shown in Figure 10-17. The corresponding interrupt sources are described in Table 10-19.

The IER, ISR, and ICR are described in more detail in the following subsections.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | — | | | | | | | | FFE_CNT | DF_CNT | DI_CNT | DO_CNT | — | | | ITO |
| Subfield | | | | | | | | | | | | | | | | |
| Reset | 0x0000 | | | | | | | | | | | | | | | |
| R/W | R/W(Interrupt Enable) R(Interrupt Status) W(Interrupt Clear) | | | | | | | | | | | | | | | |
| Addr | 0x3_1008(Interrupt Enable) 0x3_1010(Interrupt Status) 0x31018(Interrupt Clear) | | | | | | | | | | | | | | | |

| | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | — | | | | DONE Overflow | | | | CHN_4 | | CHN_3 | | CHN_2 | | CHN_1 | |
| Subfield | | | | | CH4 | CH3 | CH2 | CH1 | Err | Dn | Err | Dn | Err | Dn | Err | Dn |
| Reset | 0x0000 | | | | | | | | | | | | | | | |
| R/W | R/W(Interrupt Enable) R(Interrupt Status) W(Interrupt Clear) | | | | | | | | | | | | | | | |
| Addr | 0x3_100A(Interrupt Enable) 0x3_1012(Interrupt Status) 0x3101A(Interrupt Clear) | | | | | | | | | | | | | | | |

| | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | — | | | | CRCU | | KEU | | — | | PKEU | | — | | RNG | |
| Subfield | | | | | Err | Dn | Err | Dn | | | Err | Dn | | | Err | Dn |
| Reset | 0x0000 | | | | | | | | | | | | | | | |
| R/W | R/W(Interrupt Enable) R(Interrupt Status) W(Interrupt Clear) | | | | | | | | | | | | | | | |
| Addr | 0x3_100C(Interrupt Enable) 0x3_1014(Interrupt Status) 0x3101C(Interrupt Clear) | | | | | | | | | | | | | | | |

| | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | — | | AFEU | | — | | MDEU | | — | | AESU | | — | | DEU | |
| Subfield | | | Err | Dn | | | Err | Dn | | | Err | Dn | | | Err | Dn |
| Reset | 0x0000 | | | | | | | | | | | | | | | |
| R/W | R/W(Interrupt Enable) R(Interrupt Status) W(Interrupt Clear) | | | | | | | | | | | | | | | |
| Addr | 0x3_100E(Interrupt Enable) 0x3_1016(Interrupt Status) 0x3101E(Interrupt Clear) | | | | | | | | | | | | | | | |

**Figure 10-17. Interrupt Enable, Interrupt Status, and Interrupt Clear Registers**

**Table 10-19. Field Names in Interrupt Enable, Interrupt Status, and Interrupt Clear Registers**

| Bits | Name | Description |
|---|---|---|
| 0–7 | — | Reserved |
| 8 | FFE_CNT | Fetch FIFO enqueue count rollover<br>0 No rollover.<br>1 Fetch FIFO enqueue counter rolled over to zero (see Section 10.4.3.1.1, “Fetch FIFO Enqueue Counter”). |

### Table 10-19. Field Names in Interrupt Enable, Interrupt Status, and Interrupt Clear Registers (continued)

| Bits | Name | Description |
|---|---|---|
| 9 | DF_CNT | Descriptor Finished Count Rollover<br>0 No rollover.<br>1 The Descriptor Finished Counter rolled over to zero (see Section 10.4.3.1.2, “Descriptor Finished Counter”). |
| 10 | DI_CNT | Data In Count Rollover<br>0 No rollover.<br>1 The Data In Counter rolled over to zero (see Section 10.4.3.1.2, “Descriptor Finished Counter”). |
| 11 | DO_CNT | Data Out Count Rollover<br>0 No rollover.<br>1 The Data Out Counter rolled over to zero (see Section 10.4.3.1.2, “Descriptor Finished Counter”). |
| 12–14 | — | Reserved |
| 15 | ITO | Internal Time Out<br>0 No internal time out<br>1 an internal time out was detected<br>**Note:** Internal time out is an indication that a channel or EU has failed to respond to a slave read or write within 16 cycles, which would only occur in an impending hang condition. Assertion of this interrupt indicates the SEC controller has completed the transaction to avoid a hang—however the ’completed’ transaction does not result in a successful read or write, and the interrupt advises the system that the slave transaction was unsuccessful. |
| 20–23 | Done Overflow | Done Overflow (one bit for each channel—CH1 to CH4)<br>0 No done overflow<br>1 Done overflow error. Indicates that more than 15 Done interrupts were queued from the associated channel without a corresponding interrupt clear from the host. |
| 24–31 | Err and Dn bits for channels (CHN_1 to CHN_4) | Err<br>0 No error detected.<br>1 Error detected. Indicates that channel status register must be read to determine exact cause of the error.<br>Dn<br>0 Not DONE.<br>1 DONE bit indicates that the corresponding channel has completed a descriptor. |

Table 10-19. Field Names in Interrupt Enable, Interrupt Status, and Interrupt Clear Registers (continued)

| Bits | Name | Description |
|---|---|---|
| 36–37,<br>38-39,<br>42–43,<br>46–47,<br>50–51,<br>54–55,<br>58–59,<br>62–63 | Err and Dn bits for execution units (CRCU,KEU, PKEU, RNG, AFEU, MDEU, AESU,DEU) | Err<br>0 No error detected.<br>1 Error detected. Indicates that execution unit status register must be read to determine exact cause of the error.<br>Dn<br>0 Not Done<br>1 DONE bit indicates that the corresponding EU has completed its operation. This means that final values are available from EU registers. For EUs with output FIFOs, it means that all text data output has been placed in the output FIFO. For EUs that provide context out through the output FIFO, the EU places the context in the output FIFO after asserting PRI_DONE. |
| 0–9,<br><br>16–19,<br>32–35,<br>40–41,<br>44–45,<br>48–49,<br>52–53,<br>56–57,<br>60–61 | — | Reserved, must be cleared. |

#### 10.5.4.2.1 Interrupt Enable Register (IER)

Interrupt sources can be individually enabled by setting the corresponding IER bits (see Table 10-19 for the correspondence between IER bits and interrupt sources). If an IER bit is set, the corresponding interrupt source value is captured in the corresponding interrupt status register (ISR) bit. If an IER bit is cleared, the corresponding ISR bit remains cleared.

At reset, all IER bits are cleared, so all interrupts are disabled.

**NOTE**

For normal operation the IER should be programmed with the value 0x0031_0fff_0000_0000, which enables all channel interrupts and disables interrupts from the EUs. The EU interrupt bits are provided as a convenience during debug: during normal operation, an EU error causes the channel using that EU to generate the appropriate interrupt to the host.

#### 10.5.4.2.2 Interrupt Status Register (ISR)

Each ISR bit shows the status of a corresponding interrupt source (see Table 10-19 for the correspondence between ISR bits and interrupt sources). However, if the corresponding IER bit is cleared, then the ISR bit remains cleared.

ISR bits are cleared either by reset, or by setting the corresponding bits in the ISR or interrupt clear register.

### 10.5.4.3 Interrupt Clear Register (ICR)

The ICR provides a means of clearing the interrupt status register (ISR). Setting an ICR bit clears the corresponding bit in the ISR, and negates the interrupt output signal (assuming that particular ISR bit is the only interrupt source). When set, an ICR bit is cleared automatically on the following cycle.

**NOTE**

If the cause of an interrupt is not removed, then the ISR bit is set (and corresponding interrupt output signal asserted) a few cycles after it has been cleared using the ICR.

For this reason, the ICR is ineffective in clearing the RNG Done bit (bit 47) in the ISR. The user should use the IER to mask the RNG Done interrupt. To determine whether a descriptor-based RNG request is complete, the user should rely on Channel Done interrupts.

### 10.5.4.4 ID Register

The read-only ID register, displayed in Figure 10-18, contains the same value as the IP block revision register (see Section 10.5.4.5 below). This register provides the IP block revision information at a legacy location for software convenience.

Offset 0x3_1020 Access: Read only

| | 0 | | | | | | | | | | | | | | | 63 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | ID | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 0 0 0 | 0 0 0 0 | 0 0 1 1 | 0 0 0 0 | 0 0 0 0 | 0 0 1 1 | 0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 0 0 0 0 |

**Figure 10-18. ID Register**

### 10.5.4.5 IP Block Revision Register

The read-only IP block revision register, displayed in Figure 10-19, contains a 64-bit value that uniquely identifies the version of the SEC.

Offset 0x 3_1BF8 Access: Read only

| | 0 | | 31 | 32 | | | 63 |
|---|---|---|---|---|---|---|---|
| R | IP_ID | IP_MJ | IP_MN | — | IP_INT | — | IP_CFG |
| W | | | | | | | |
| Reset | All zeros | | | | | | |

**Figure 10-19. IP Block Revision Register**

Table 10-20 describes the fields of the IP block revision register.

**Table 10-20. IP Block Revision Register Fields**

| Bits | Name | Description |
|---|---|---|
| 0–15 | IP_ID | IP block identifier. This field value is currently set as 0x0030 |
| 16-23 | IP_MJ | IP major revision number. This field value is currently set as 0x03. |
| 24-31 | IP_MN | IP minor revision number. This field value is currently set as 0x00 |
| 32-39 | — | Reserved |
| 40-47 | IP_INT | IP block integration options. Field value depends on the options of the specific SoC |
| 48-55 | — | Reserved |
| 56-63 | IP_CFG | IP block configuration options. Field value depends on the options of the specific SoC |

### 10.5.4.6 Master Control Register (MCR)

The MCR, shown in Figure 10-20, controls certain functions in the controller and provides a means for software to reset the SEC. Table 10-21 describes the MCR fields.

Offset 0x3_1030 Access: Read/Write

| | 0 … 21 | 22 … 23 | 24 … 29 | 30 | 31 |
|---|---|---|---|---|---|
| R<br>W | — | PRIORITY | — | GIH | SWR |
| Reset | All zeros | | | | |

| | 32 … 39 | 40 … 47 | 48 … 55 | 56 … 63 |
|---|---|---|---|---|
| R<br>W | CHN3_EU_PR_CNT | CHN4_EU_PR_CNT | CHN3_BUS_PR_CNT | CHN4_BUS_PR_CNT |
| Reset | All zeros | | | |

**Figure 10-20. Master Control Register**

**Table 10-21. Master Control Register Fields**

| Bits | Name | Description |
|---|---|---|
| 0–21 | — | Reserved |
| 22-23 | Priority | Priority on Master Bus. The setting of these bits determines the transaction priority level the SEC asserts to the SoC's internal arbiter. The SEC does not dynamically alter its priority level based on system congestion or SEC utilization; however, software may change the SEC priority level in real time.<br>00 Lowest Priority (default)<br>01 Next Lowest Priority<br>10 Next Highest Priority<br>11 Highest Priority |
| 24-29 | — | Reserved |

**Table 10-21. Master Control Register Fields (continued)**

| Bits | Name | Description |
|---|---|---|
| 30 | GIH | Global Inhibit. Setting this bit indicates that SoC master bus transfers are defined as not snoopable and results in lowering the snoop attribute of bus requests generated by the external gasket (see note following table).<br>0 SoC master bus transfers are defined as snoopable (default)<br>1 SoC master bus transfers are defined as not snoopable |
| 31 | SWR | Software Reset. Setting this bit causes a global software reset. Upon completion of the reset, this bit is automatically cleared.<br>0 Do not reset<br>1 Global reset |
| 32–39 | CHN3_EU_PR_CNT | Channel 3 EU Priority Count. In weighted priority arbitration, this field gives the number of times that Channel 3 is denied access for a requested EU before its priority is elevated (see Section 10.5.2.2, “Weighted Priority Arbitration”).<br>**Note:** If both CHN3_EU_PR_CTR and CHN4_EU_PR_CTR are zero, the controller assigns EU’s on a pure round-robin basis. If either of these counters is zero and the other is non-zero, then the zero is interpreted as 256. |
| 40–47 | CHN4_EU_PR_CNT | Channel 4 EU Priority Count. In weighted priority arbitration, this field gives the number of times that Channel 4 is denied access for a requested EU before its priority is elevated (see Section 10.5.2.2, “Weighted Priority Arbitration”).<br>**Note:** If both CHN3_EU_PR_CTR and CHN4_EU_PR_CTR are zero, the controller assigns EU’s on a pure round-robin basis. If either of these counters is zero and the other is non-zero, then the zero is interpreted as 256. |
| 48–55 | CHN3_BUS_PR_CNT | Channel 3 Bus Priority Count. In weighted priority arbitration, this field gives the number of times that Channel 3 is denied access to the polychannel before its priority is elevated (see Section 10.5.2.2, “Weighted Priority Arbitration”).<br>**Note:** If both CHN3_BUS_PR_CTR and CHN4_BUS_PR_CTR are zero, the controller assigns the polychannel on a pure round-robin basis. If either of these counters is zero and the other is non-zero, then the zero is interpreted as 256. |
| 56–63 | CHN4_BUS_PR_CNT | Channel 4 Bus Priority Counter. In weighted priority arbitration, this field gives the number of times that Channel 4is denied access to the polychannel before its priority is elevated (see Section 10.5.2.2, “Weighted Priority Arbitration”).<br>If both CHN3_BUS_PR_CTR and CHN4_BUS_PR_CTR are zero, the controller assigns the polychannel on a pure round-robin basis. If either of these counters is zero and the other is non-zero, then the zero is interpreted as 256. |

## NOTE

By default, All SEC memory transactions are snooped by the coherency module of the MPC85*xx*. This is part of the wiring of the SEC interface and requires no user intervention. Bit 30 in the MCR is used to inhibit cache snooping of SEC transactions in non-MPC85*xx* situations.

## 10.6 Power Saving Mode

The SEC may be disabled by setting DEVDISR[SEC] in the SoC. The clocks to the SEC are active by default. The SEC should not be enabled/disabled during normal operation.

SEC disablement is delayed if the disable request is made while descriptors are being processed. Once notified of the disable request, the SEC channels complete their current tasks, and then are forced to idle (with no additional reads from the fetch descriptor FIFO). Once all channels are idle, then SEC permits disablement.

## 10.7 Execution Units

Execution unit (EU) is the term used for a functional block that performs the mathematical manipulations required by cryptographic processing. The following execution units are used in the SEC (covered here in alphabetical order):

- Advanced Encryption Standard Execution Unit (AESU) implementing the Rijndael symmetric key cipher.
- ARC4 Execution Unit (AFEU)
- Cyclical Redundancy Check Unit (CRCU)
- Data Encryption Standard Execution Unit (DEU)
- Kasumi (f8/f9) Execution Unit (KEU)
- Message Digest Execution Unit (MDEU)
- Public Key Execution Unit (PKEU)
- Random Number Generator Unit(RNGU)

Working together, the EUs can perform high-level cryptographic tasks, such as IPsec Encapsulating Security Protocol (ESP) and digital signature. The remainder of this chapter provides details about these execution units, including modes of operation, status and control registers, and FIFOs.

### 10.7.1 Advanced Encryption Standard Execution Unit (AESU)

This section contains details about the Advanced Encryption Standard Execution Unit (AESU), including modes of operation, status and control registers, and FIFOs.

**NOTE**

Most of the registers described in this section are not accessed by the host under normal operation. They are documented here mainly for debug purposes. Normally the AESU is used through channel-controlled access, so that most reads and writes of AESU registers are directed by the SEC channels. Driver software performs host-controlled register accesses only on a few registers for initial configuration and error handling.

### 10.7.1.1 ICV Checking in AESU

For CCM, GCM, CMAC (OMAC1), and XCBC-MAC cipher modes, the AESU includes an ICV checking feature which can generate an ICV and compare it to another supplied ICV.

There are two methods for returning the pass/fail result of ICV checking to the host:

- The ICV check result can be sent to the host by a writeback of EU status fields into host memory. This is enabled as follows:
  - Set either the IWSE or AWSE bit in the channel configuration register (see Section 10.4.4.1, “Channel Configuration Register (CCR)”)
  - Set the ICE bit in the interrupt mask register (Section 10.7.1.8, “AESU Interrupt Mask Register”).

  In this case the normal done signaling (by interrupt or writeback) is undisturbed.
- The ICV checking result can be sent to the host by interrupt. This is enabled as follows:
  - Clear the ICE bit in the interrupt mask register
  - Clear both IWSE and AWSE bits in the channel configuration register.

  In this case, then the normal done signaling (by interrupt or writeback) occurs if there is no ICV mismatch. If an ICV mismatch occurs, then an error interrupt is sent to the host, but no channel done interrupt or writeback.

### 10.7.1.2 AESU Mode Register

The AESU mode register contains 11 non-reserved bits which are used to program the AESU. The mode register is cleared when the AESU is reset or re-initialized. Setting a reserved AESU mode register bit generates a data error. If the mode register is modified during processing, a context error is generated.

Figure 10-21 shows the AESU mode register, and Table 10-22 describes its fields. In normal operation, the register’s values are set by the descriptor header (see Section 10.3.2, “Descriptor Format: Header Dword”).

Offset 0x3_4000 Access: Read/Write

| | 0–49 | 50–52 | 53–55 | 56–57 | 58 | 59 | 60 | 61–62 | 63 |
|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | SCM | — | ECM | AUX2 | AUX1 | AUX0 | CM | ED |
| Reset | All zeros | | | | | | | | |

**Figure 10-21. AESU Mode Register**

**Table 10-22. AESU Mode Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–49 | — | Reserved |
| 50–52 | SCM | Sub-Cipher Mode. Specifies additional options specific to particular cipher modes.<br>• XOR cipher mode: specifies the number of sources to be XORed together. Valid values are 2-6. For all other cipher modes, this field must be 0. |

**Table 10-22. AESU Mode Register Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 53–55 | — | Reserved, must be cleared. |
| 56–57 | ECM | Extended Cipher Mode. Used in combination with bits 61:62 (Cipher Mode) to select the cipher mode for AES operation. See Table 10-23 on page 10-60 for mode bit combinations. |
| 58 | AUX2 | AUX2 Mode. Definition depends upon the value of the 4 Cipher Mode (CM) and Extended Cipher Mode (ECM) bits: |
| | AUX2 = Finalize MAC for CCM and GCM modes | • CCM and GCM Cipher Modes (ECM=1X, CM=0X): Generate Final MAC Bit—Processes final message block and generates final MAC tag at the end of message processing.<br>0 = Do not generate final MAC tag<br>1 = Generate final MAC tag after CCM/GCM processing is complete. Note that for GCM, when message processing is split into multiple descriptors, it must be AUX1=1 when AUX2=1. |
| | AUX2 = ICV Bit | • XCBC-MAC and CMAC Cipher Modes (ECM=10, 01, CM=10): ICV Bit—Enables XCBC-MAC with ICV and CMAC with ICV Cipher Modes<br>0 = XCBC-MAC or CMAC cipher mode<br>1 = XCBC-MAC with ICV or CMAC with ICV cipher mode |
| | AUX2 = Enable RBP | • CBC, CBC-RBP Cipher Modes (ECM=00, CM=01): RBP Bit—Enables CBC-RBP<br>0 = CBC cipher mode<br>1 = CBC-RBP cipher mode |
| 59 | AUX1 | AUX1 Mode. Definition depends upon the value of the 4 Cipher Mode and Extended Cipher Mode bits: |
| | AUX1 = Initialize CCM | • CCM Cipher Mode (ECM=10, CM=00): Initialize Mode Bit—Initializes AESU for new message<br>0 = Do not initialize (context is loaded by host)<br>1 = Initialize new message with nonce/initialization vector |
| | AUX1 = Generate Final GHASH | • GCM Cipher Mode (ECM=10, CM=01): Generate Final GHASH Bit—Enables completion of GHASH computation by signaling that the last iteration of GHASH should be performed. This last iteration performs XOR of the current (intermediate) GHASH result with the concatenation of additional authenticated data (AAD) and ciphertext bit lengths in case of GHASH(H, AAD, ciphertext), or with the concatenation of $0^{64}$ and the bit length of IV in case of GHASH(H, {}, IV). As an exception, this bit should be cleared if the whole message (IV+AAD+text data) together with the generation of the final MAC is processed with one descriptor since in that case the generation of final GHASH is implied. Incidentally, whenever AUX1=1 in GCM cipher mode, the total bit lengths of AAD, text data or IV must be provided in context registers 9-10.<br>0 = Do not perform the last iteration in GHASH(H, AAD, ciphertext) or GHASH(H, {}, IV) unless the message is processed and the final MAC computed in 1 descriptor.<br>1 = Generate the final result of GHASH(H, AAD, ciphertext) or GHASH(H, {}, IV)—implies that the message processing is split into multiple descriptors. |
| | AUX1 = Use Context for XCBC-MAC derived keys | • XCBC-MAC Cipher Mode (ECM=10, CM=10): Load Keys—Do not compute K1, K2 and K3, but instead use the keys loaded in the Key Data Registers (K1), and Context Registers 5-6 (K2) and 7-8 (K3).<br>0 = Compute K1=E(K, 16{01}), K2=E(K, 16{02}), K3=E(K, {03}) and write K1 to Context Registers 3-4, K2 to 5-6, and K3 to 7-8.<br>1 = Load keys: K1= [Key Data Reg 1-2], K2= [Reg 5-6], K3=[Reg 7-8] |
| | AUX1 = Use Context for CMAC derived keys | • CMAC Cipher Mode (ECM=01, CM=10): Load Keys—Do not compute E(K, $0^{128}$) to derive K1 and K2, but instead use the value loaded in Context Registers 3-4. This is useful after a context switch. Deriving K1 and K2 does not incur any timing penalty.<br>0 = Compute E(K, $0^{128}$) and write it to Context Registers 3-4<br>1 = Load E(K, $0^{128}$) and preserve it in Context Registers 3-4 |

**Table 10-22. AESU Mode Register Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 60 | AUX0<br>AUX0 = GCM GHASH Only<br>AUX0 = Finalize Mac for XCBC-MAC and CMAC modes | AUX0 Mode. Definition depends upon the value of the 4 Cipher Mode and Extended Cipher Mode bits, and Encrypt/Decrypt bit:<br>• GCM Cipher Mode (ECM=10, CM=01) and Encrypt (ED=1): Specifies GHASH mode—performs GHASH on AAD and ciphertext<br>0 = Perform GCM encryption<br>1 = Compute GHASH(H, AAD, ciphertext)<br>• XCBC-MAC, CMAC Cipher Modes (ECM=10, 01, CM=10): Do Not Generate Final MAC Bit—Does not generate final MAC tag at the end of message processing (used only when splitting a message into multiple descriptors)<br>0 = Generate final MAC tag by XORing the final data block with K2/K3 (for XCBC-MAC) or K1/K2 (for CMAC) before encryption<br>1 = Do not generate final MAC tag by XORing final data block before encryption. This enables message processing to be interrupted on the block boundary and later continued after a context switch. |
| 61–62 | CM | Cipher Mode. Used in combination with bits 56:57 (Extended Cipher Mode) to select the cipher mode for AES operation. See Table 10-23 for mode bit combinations. |
| 63 | ED | Encrypt/Decrypt. If set, AESU operates the encryption algorithm; if cleared, AESU operates the decryption algorithm.<br>0 Perform decryption<br>1 Perform encryption<br>**Note:** This bit is ignored in CTR, SRT, CMAC, and XCBC-MAC cipher modes. |

Table 10-23 shows the AESU field settings corresponding to different AES cipher modes.

**Table 10-23. AES Cipher Modes**

| Cipher Mode | ECM (56:57) | AUX2 (58) | CM (61:62) |
|---|---|---|---|
| ECB | 00 | X | 00 |
| CBC | 00 | X | 01 |
| CBC-RBP | 00 | 1 | 01 |
| OFB | 00 | X | 10 |
| CTR | 00 | X | 11 |
| LRWXTS | 01 | X | 01 |
| CMAC | 01 | X | 10 |
| CMAC with ICV | 01 | 1 | 10 |
| SRT[1] | 01 | X | 11 |
| CCM | 10 | X | 00 |
| GCM | 10 | X | 01 |
| XCBC-MAC | 10 | 0 | 10 |
| XCBC-MAC with ICV | 10 | 1 | 10 |
| CFB128 | 10 | X | 11 |

Table 10-23. AES Cipher Modes (continued)

| Cipher Mode | ECM (56:57) | AUX2 (58) | CM (61:62) |
|---|---|---|---|
| CCM with ICV | 11 | X | 00 |
| GCM with ICV | 11 | X | 01 |
| XOR | 11 | X | 11 |
| Reserved | all others | | |

[1] SRT is not a new AES cipher mode, it is an AESU method of performing AES counter mode with reduced context loading overhead specifically for performing SRTP. It should be used with descriptor type 0010_1 'srtp' (but may also be used with descriptor type 0010_0 for IPsec with AES counter mode). See the section on "Context for SRT Cipher Mode" for more information on how SRT cipher mode reduces context loading overhead.

### 10.7.1.3 AESU Key Size Register

The AESU key size register, shown in Figure 10-22, is used to specify the number of bytes in the key (16, 24, or 32). Any key data beyond the number of bytes specified in the key size register is ignored. This register is cleared when the AESU is reset or re-initialized. If a key size other than 16, 24, or 32 bytes (or other than 16 bytes, in XCBC-MAC cipher mode) is specified, an illegal key size error is generated. If the key size register is modified during processing, a context error is generated.

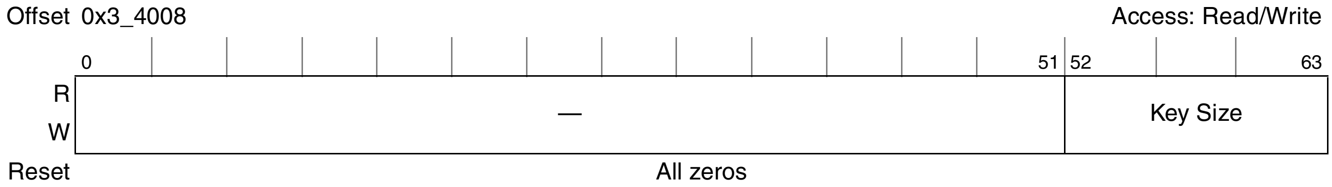


Figure 10-22. AESU Key Size Register

### 10.7.1.4 AESU Data Size Register

The AESU data size register, shown in Figure 10-23, is used to specify the number of bits (not bytes) of plaintext/ciphertext to be processed in the current descriptor. The number of data size register bits used by the SEC, and the acceptable values for these bits, vary depending on the AES cipher mode selected as specified in Table 10-24.

Writing to this register signals the AESU to start processing data from the input FIFO as soon as it is available. If the value of data size is modified during processing, a context error is generated. The register is cleared when the AESU is reset or re-initialized.

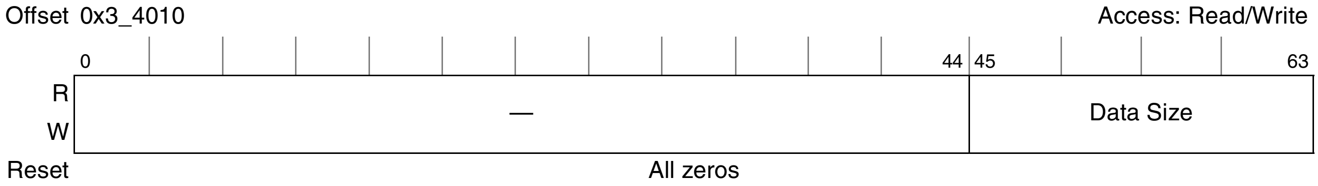


Figure 10-23. AESU Data Size Register

Table 10-24. Use of Data Size Register

| AESU Cipher Mode | Register bits used by SEC (others are don't cares) | Legal Values (data size in bits) |
|---|---|---|
| ECB, CBC | lowest 7 bits [57:63] | must be a multiple of 128 |
| OFB, CMAC, SRT, CCM, XCBC-MAC, CFB128 |  | must be a multiple of 8 |
| XTS | all bits | must be a multiple of 8, minimum 128 |
| GCM | all bits | any value |
| XOR | all bits | must be a multiple of 256 |

### 10.7.1.5 AESU Reset Control Register

The AESU reset control register has three self-clearing bits, where each bit corresponds to a different type of AESU reset. Figure 10-24 shows the AESU reset control register, and Table 10-25 describes its fields.

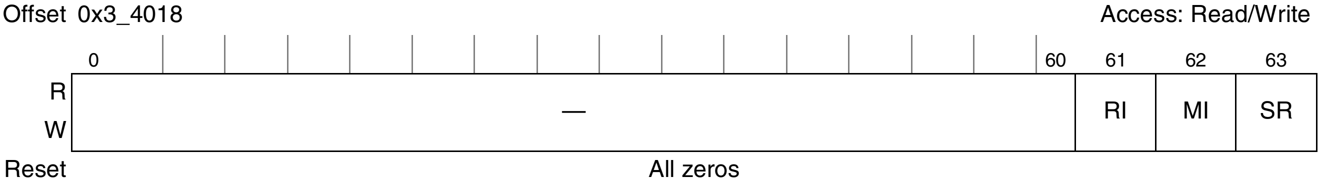


Figure 10-24. AESU Reset Control Register

Table 10-25. AESU Reset Control Register Field Descriptions

| Bits | Names | Description |
|---|---|---|
| 0–60 | — | Reserved |
| 61 | RI | Reset Interrupt. Setting this bit resets the AESU's done and error interrupts, and resets the state of the AESU interrupt status register.<br>0 Do not reset<br>1 Reset interrupt logic |
| 62 | MI | Module Initialization. The same as software reset (including the initialization routine: see SR bit description below), except that the interrupt mask register remains unchanged.<br>0 Do not reset<br>1 Reset most of AESU |
| 63 | SR | Software reset. Functionally equivalent to hardware reset, but applies only to AESU. All registers and internal state are returned to their defined reset states. The RESET_DONE bit in the AESU status register indicates when this initialization routine is complete<br>0 Do not reset<br>1 Full AESU reset |

### 10.7.1.6 AESU Status Register

The AESU status register is a read-only register that reflects the state of six status outputs. Writing to this location results in an address error being reflected in the AESU interrupt status register.

Figure 10-25 shows the AESU status register, and Table 10-26 describes its fields.

Offset 0x3_4028 Access: Read only

| | 0–39 | 40–47 | 48–55 | 56–57 | 58 | 59–60 | 61 | 62 | 63 |
|---|---|---|---|---|---|---|---|---|---|
| R | — | OFL | IFL | — | HALT | ICCR | EI | DI | RD |
| W | | | | | | | | | |
| Reset | All zeros | | | | | | | | |

**Figure 10-25. AESU Status Register**

**Table 10-26. AESU Status Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–39 | — | Reserved |
| 40-47 | OFL | The number of dwords currently in the output FIFO |
| 48-55 | IFL | The number of dwords currently in the input FIFO |
| 56-57 | — | Reserved |
| 58 | HALT | Halt. Indicates that the AESU has halted due to an error.<br>0 AESU not halted<br>1 AESU halted<br>**Note:** Because the error causing the AESU to stop operating may be masked before reaching the interrupt status register, the AESU interrupt status register is used to provide a second source of information regarding errors preventing normal operation. |
| 59-60 | ICCR | Integrity Check Comparison Result<br>00 No integrity check comparison was performed.<br>01 The integrity check comparison passed.<br>10 The integrity check comparison failed.<br>11 Reserved<br>**Note:** A passed or failed result is generated only if the cipher mode with ICV checking is selected |
| 61 | EI | Error interrupt: This status bit reflects the state of the error interrupt signal, as sampled by the controller interrupt status register (Section 10.5.4.2.2, “Interrupt Status Register (ISR)”).<br>0 AESU is not signaling error<br>1 AESU is signaling error |
| 62 | DI | Done interrupt: This status bit reflects the state of the done interrupt signal, as sampled by the controller interrupt status register (Section 10.5.4.2.2, “Interrupt Status Register (ISR)”).<br>0 AESU is not signaling done<br>1 AESU is signaling done |
| 63 | RD | Reset Done. This status bit, when high, indicates that AESU has completed its reset sequence, as reflected in the signal sampled by the appropriate channel.<br>0 Reset in progress<br>1 Reset done<br>**Note:** This bit resets to 0 but has typically switched to 1 by the time a user checks the register, indicating the EU is ready for operation. |

#### 10.7.1.7 AESU Interrupt Status Register

The AESU interrupt status register indicates which unmasked errors have occurred and have generated error interrupts to the channel. Each bit in this register can only be set if the corresponding bit of the AESU interrupt mask register is zero (see Section 10.7.1.8, “AESU Interrupt Mask Register”). If an AESU interrupt mask register bit is set, the corresponding AESU interrupt status bit is always zero regardless of the error status.

If the AESU interrupt status register is non-zero, the AESU halts and the AESU error interrupt signal is asserted to the controller (see Section 10.5.4.2.2, “Interrupt Status Register (ISR)”). In addition, if the AESU is being operated through channel-controlled access, then an interrupt signal is generated to the channel to which this EU is assigned. The EU error then appears in bit 55 of the channel status register (see Table 10-15) and generates a channel error interrupt to the controller.

Interrupt status register bits can be set by writes from the host, but only if the corresponding bit is cleared in the interrupt mask register. Bits masked by the interrupt mask register bits are always zero.

The AESU interrupt status and interrupt mask registers can be cleared by programming the AESU reset control register, as described in Section 10.7.1.5, “AESU Reset Control Register”.

The AESU interrupt status register fields are shown in Figure 10-26. These fields are described in Table 10-27.

Offset 0x3_4030 Access: Read only

| | 0–48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 63 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | | ICE | — | IE | ERE | CE | KSE | DSE | ME | AE | OFE | IFE | — | IFO | OFU | — | ICE |
| W | | | | | | | | | | | | | | | | | |
| Reset | All zeros | | | | | | | | | | | | | | | | |

**Figure 10-26. AESU Interrupt Status Register**

**Table 10-27. AESU Interrupt Status Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–48 | — | Reserved |
| 49 | ICE | Integrity Check Error:<br>0 No error detected<br>1 Integrity check error detected. An ICV check was performed and the supplied ICV did not match the one computed by the AESU. |
| 50 | — | Reserved |
| 51 | IE | Internal Error. An internal processing error was detected while the AESU was processing.<br>0 No error detected<br>1 Internal error<br>**Note:** This bit is asserted any time an enabled error condition occurs and can only be cleared by setting the corresponding bit in the interrupt mask register or by resetting the AESU. |

### Table 10-27. AESU Interrupt Status Register Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 52 | ERE | Early Read Error. An AESU context register was read while the AESU was processing.<br>0 No error detected<br>1 Early read error |
| 53 | CE | Context Error. An AESU key register or the key size register, data size register, mode register, or IV register was modified while AESU was processing<br>0 No error detected<br>1 Context error |
| 54 | KSE | Key Size Error. An inappropriate value (not 16, 24 or 32 bytes) was written to the AESU key size register.<br>0 No error detected<br>1 Key size error |
| 55 | DSE | Data Size Error (DSE): A value was written to the AESU data size register that is not a proper size. See Section 10.7.1.4, “AESU Data Size Register.”<br>0 No error detected<br>1 Data size error |
| 56 | ME | Mode Error. Indicates that invalid data was written to a register or a reserved mode bit was set.<br>0 Valid Data<br>1 Reserved or invalid mode selected |
| 57 | AE | Address Error. An illegal read or write address was detected within the AESU address space.<br>0 No error detected<br>1 Address error |
| 58 | OFE | Output FIFO Error. The AESU output FIFO was detected non-empty upon write of AESU data size register.<br>0 No error detected<br>1 Output FIFO non-empty error |
| 59 | IFE | Input FIFO Error. The AESU input FIFO was detected non-empty upon generation of done interrupt.<br>0 No error detected<br>1 Input FIFO non-empty error |
| 60 | — | Reserved |
| 61 | IFO | Input FIFO Overflow. The AESU Input FIFO was pushed while full.<br>0 No error detected<br>1 Input FIFO has overflowed<br>**Note:** When operated through channel-controlled access, the SEC implements flow control, and FIFO size is not a limit to data input. When operated through host-controlled access, the AESU cannot accept FIFO inputs larger than 256 bytes without overflowing. |
| 62 | OFU | Output FIFO Underflow. The AESU Output FIFO was read while empty.<br>0 No error detected<br>1 Output FIFO has underflow error |
| 63 | — | Reserved |

### 10.7.1.8 AESU Interrupt Mask Register

The AESU interrupt mask register, controls the setting of bits in the AESU interrupt status register, as described in Section 10.7.1.7, “AESU Interrupt Status Register”. If an AESU interrupt mask register bit is set, then the corresponding interrupt status register bit is always zero.

As shown in Figure 10-27, the interrupt mask register has the same field designations as the interrupt status register. Table 10-28 describes the AESU interrupt mask register fields.

Offset 0x3_4038 Access: Read/Write

| | 0–48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | ICE | — | IE | ERE | CE | KSE | DSE | ME | AE | OFE | IFE | — | IFO | OFU | — |
| Reset | 1000 | | | | | | | | | | | | | | | |

**Figure 10-27. AESU Interrupt Mask Register**

**Table 10-28. AESU Interrupt Mask Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–48 | — | Reserved |
| 49 | ICE | Integrity Check Error. The supplied ICV did not match the one computed by the AESU.<br>0 Integrity check error enabled.<br>1 Integrity check error disabled<br>**Note:** ICE should not be enabled if using EU status writeback (see bits IWSE and AWSE in Section 10.4.4.1, “Channel Configuration Register (CCR)”). |
| 50 | — | Reserved |
| 51 | IE | Internal Error. An internal processing error was detected while the AESU was processing.<br>0 Internal error enabled<br>1 Internal error disabled |
| 52 | ERE | Early Read Error. An AESU context register was read while the AESU was processing.<br>0 Early read error enabled<br>1 Early read error disabled |
| 53 | CE | Context Error. An AESU key register or the key size register, data size register, mode register, or IV register was modified while the AESU was processing.<br>0 Context error enabled<br>1 Context error disabled |
| 54 | KSE | Key Size Error. An inappropriate value (non 16, 24 or 32 bytes) was written to the AESU key size register<br>0 Key size error enabled<br>1 Key size error disabled |
| 55 | DSE | Data Size Error. Indicates that the number of bits to process is out of range.<br>0 Data size error enabled<br>1 Data size error disabled |
| 56 | ME | Mode Error. Indicates that invalid data was written to a register or a reserved mode bit was set.<br>0 Mode error enabled<br>1 Mode error disabled |

**Table 10-28. AESU Interrupt Mask Register Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 57 | AE | Address Error. An illegal read or write address was detected within the AESU address space.<br>1 Address error disabled<br>0 Address error enabled |
| 58 | OFE | Output FIFO Error. Indicates the AESU Output FIFO was detected non-empty upon write of AESU data size register<br>0 Output FIFO non-empty error enabled<br>1 Output FIFO non-empty error disabled |
| 59 | IFE | Input FIFO Error. Indicates he AESU Input FIFO was detected non-empty upon generation of done interrupt<br>0 Input FIFO non-empty error enabled<br>1 Input FIFO non-empty error disabled |
| 60 | — | Reserved |
| 61 | IFO | Input FIFO Overflow. Indicates the AESU Input FIFO was pushed while full.<br>0 Input FIFO overflow error enabled<br>1 Input FIFO overflow error disabled |
| 62 | OFU | Output FIFO Underflow. Indicates the AESU output FIFO was read while empty.<br>0 Output FIFO underflow error enabled<br>1 Output FIFO underflow error disabled |
| 63 | — | Reserved |

### 10.7.1.9 AESU ICV Size Register

The ICV size register, shown in Figure 10-28, is used in AES hashing modes CMAC and GCM to specify the number of most significant bytes in the received MAC tag supplied in context registers 3–4. AES truncates the computed MAC in context registers 1–2 to the same number of bytes, and writes zeros in the remaining LSB’s. It follows that the received MAC can be padded to 16 bytes with arbitrary data (not necessarily zeros) when written into context registers 3–4. Acceptable values for ICV size are 8, 10, 12, 14 and 16 bytes in CMAC, or 8, 12, and 16 bytes in GCM. All other sizes are interpreted as 16.

In XCBC-MAC cipher mode, the ICV size register is not used. The received MAC (written to context registers 9-10) is always truncated to the most significant 12 bytes, as defined in the XCBC-MAC-96 for IPsec specification. The computed MAC written at the end of processing to Context Registers 1-2 is a full 16-byte MAC.

In CCM mode with ICV, the ICV size register is not used. Instead, the tag size is encoded within one of the CCM formatting flags.

Offset 0x3_4040 Access: Read/Write

| | 0–56 | 57–63 |
|---|---|---|
| R<br>W | — | ICV Size |
| Reset | All zeros | |

**Figure 10-28. AESU ICV Size Register**

### 10.7.1.10 AESU End of Message Register

The AESU end of message register, shown in Figure 10-29, is used to signal to the AESU that the final message block has been written to the input FIFO (in channel-driven access, this signaling is done automatically). The AESU will not process the last block of data in its input FIFO until this register is written. Once the end of message register is written, the AESU processes any remaining data in the input FIFO and generates the done interrupt.

The value written to this register does not matter: ordinarily, zero is written. A read of this register always returns a zero value.

Offset 0x3_4050 Access:Write only

| | 0 – 63 |
|---|---|
| R | |
| W | AESU End of Message |
| Reset | All zeros |

**Figure 10-29. AESU End of Message Register**

### 10.7.1.11 AESU Context Registers

There are twelve 64-bit context data registers that allow the host to read/write the contents of the context used to process a message. The context must be written prior to the key data. If the context registers are written during message processing, a context error is generated. All context registers are cleared when an initialization or a hard or soft reset is performed.

If a message is processed through the AESU in two separate operations (that is, using two descriptors), then the context must be read from the SEC at the end of the first operation and then restored at the beginning of the second operation.

Context is always read and restored as a contiguous subset of the twelve context registers ending with the highest numbered register used in that cipher mode. For example, when restoring context in CTR cipher mode (which uses context registers 5–7, as shown in Table 10-29), context registers 1–7 must be written (where registers 1–4 must be filled with zeros).

Context register assignments for cipher modes for confidentiality, data integrity, and combined confidentiality and integrity are described in the following subsections.

#### 10.7.1.11.1 Context for Confidentiality Cipher Modes

The context registers for the different cipher modes which provide confidentiality only are summarized in Table 10-29. The registers are described in more detail in the following subsections.

Table 10-29. AESU Context Registers for Confidentiality Modes

| Context Register (byte address) | Confidentiality-only Cipher Mode | | | | |
|---|---|---|---|---|---|
| | ECB | CBC / CBC-RBP / OFB / CFB128 | CTR | XTS | SRT |
| 1 (0x34100) | — | IV* | 0 | I* | Initial Counter Value* |
| 2 (0x34108) | | | | sector size* | |
| 3 (0x34110) | — | — | 0 | — | Counter Modulus* |
| 4 (0x34118) | | | | | — |
| 5 (0x34120) | — | — | Initial Counter Value* | — | — |
| 6 (0x34128) | | | | | |
| 7 (0x34130) | — | — | Counter Modulus Exponent* | — | — |

**Notes:**
Context Registers 8 through 12 are not used for these modes
* Must be written at start of new message, except if zero
— don’t care

### Context for ECB Mode

ECB does not use any context registers.

### Context for CBC, CBC-RBP, OFB, and CFB128 Cipher Modes

In CBC, CBC-RBP, OFB, and CFB128 cipher modes, the first two context data registers allow the host to read/write the contents of the initialization vector (IV) as follows:

- Context register 1 holds the least significant bytes of the initialization vector (bytes 1–8).
- Context register 2 holds the most significant bytes of the initialization vector (bytes 9–16).

The IV must be written prior to the message data. If the IV registers are written during message processing, or the mode is not set, a context error is generated.

The IV registers may only be read after processing has completed, as indicated by the assertion of the done interrupt (DI) bit in the AESU status register (see Section 10.7.1.6, “AESU Status Register”). If the IV registers are read prior to the assertion of DI, then an early read error is generated.

### Context for Counter (CTR) Cipher Mode

In counter cipher mode, a random 128-bit initial counter value is incremented modulo $2^M$ with each block processed. The running counter is encrypted and XORed with the plaintext to derive the ciphertext, or with the ciphertext to recover the plaintext. The modulus exponent M can be set between 8 and 128, in multiples of 8.

As shown in Table 10-29, in CTR mode context registers 5–6 hold the initial counter value, and context register 7 holds the modulus exponent M .

### $f = \alpha^{128} + \alpha^{7} + \alpha^{2} + \alpha + 1$ Context and Operation for XTS Cipher Mode

IEEE P1619 describes XTS mode as a tweakable block cipher used for encryption of sector-based storage. The key material for XTS consists of a data encryption key (used by the AES block cipher) as well as a tweak key that is used to incorporate the logical position of the data block into the encryption. The key is parsed as a concatenation of two fields of equal size called Key1 and Key2 (16 or 32 bytes each).

A 256-bit or 512-bit key is associated with an ordered sequence of sectors, numbered consecutively. The sequence of sectors that are associated with the key is related to the scope of that key. In order to encrypt or decrypt a sector, the sequence number of this sector within the scope of the key (I) must be known. Each 16-byte block within the sector has its own sequence number that gives the logical position of the data block inside the sector.

The tweak value (T) is computed based on both the sector number (I) and the 16-byte block number within the sector (j) as

$T_0$ = AES_encrypt(I, Key2);

$T_j$ = $\text{xtime}^{j}(T_0)$;

xtime(L) is defined as follows, where L is a 128-bit vector with L[127] as most significant bit:

- If L[127]=0, then xtime(L)=L<<1 (where ‘<<‘ denotes bitwise left shift)
- Else xtime(L) = (L<<1) XOR 0x87.

where L is a 128-bit vector and L[127] is it’s most significant bit. The irreducible polynomial $f = \alpha^{128} + \alpha^{7} + \alpha^{2} + \alpha + 1$ is used in the tweak calculation.

When the last block of the message is not a full 16-byte block, processing of the last two blocks implements a borrowing mechanism whereby bits from the next-to-last block ciphertext are appended to the last block plaintext to pad it out to a 16-byte boundary. This is described in detail in IEEE P1619.

For sector byte sizes that are divisible by 16, XTS mode also supports processing of multiple sectors per session where the sector sequence number (I) is automatically incremented. When multiple sectors are decrypted, the tweak key (Key2) is expanded once for the initial tweak computation pertaining to the first sector; this expanded value is directly used for other sectors. In case that a message needs to be processed in multiple XTS sessions, message splitting must be on a sector boundary.

XTS cipher mode uses context register 1 for the index (I) and context register 2 for the sector size (see Table 10-29).

For host-controlled operation of the AESU in XTS cipher mode, the following steps must be performed:

1. Reset
2. Write the Mode Register to:
   a. Set Cipher Mode to XTS
   b. Specify encryption/decryption
3. Load Key
4. Load I into context register 1 and the sector size in bytes into context register 2. Note that I must be written as big-endian (LSB in the left-most bit positions 63-56), while sector size must be in

little-endian format.

5. Set Key size
6. Set Data size
7. While available:
   a. Load plaintext (for encryption) or ciphertext (for decryption) blocks
   b. Unload ciphertext (for encryption) or plaintext (for decryption) blocks
8. Write to the end of message register
9. Unload final ciphertext (for encryption) or plaintext (for decryption) blocks

### Context for SRT Cipher Mode

As mentioned in the footnote to Table 10-23, SRT is not a new AES cipher mode but rather an AESU method of performing AES-CTR cipher mode with reduced context loading overhead specifically for performing SRTP. As with CTR cipher mode, a random 128-bit initial counter value is incremented modulo $2^M$ with each block processed. The running counter is encrypted and XORed with the plaintext to derive the ciphertext, or with the ciphertext to recover the plaintext. The modulus exponent M can be set between 8 and 128 in multiples of 8.

As shown in Table 10-29, in SRT mode context registers 1–2 hold the initial counter value, and context register 3 holds the modulus exponent M .

### 10.7.1.11.2 Context for Data Integrity Cipher Modes

The context registers for the different cipher modes which provide data integrity only are summarized in Table 10-30. The registers are described in more detail in the following subsections.

**Table 10-30. AESU Context Registers for Integrity Modes**

| Context Register # (byte address) | Cipher Mode providing only Data Integrity | | |
|---|---|---|---|
| | XCBC-MAC | GCM-GHASH | CMAC (OMAC1) |
| 1 (0x34100) | Computed MAC | Computed MAC | Computed MAC |
| 2 (0x34108) | | | |
| 3 (0x34110) | Received MAC* | | Received MAC* |
| 4 (0x34118) | | | |
| 5 (0x34120) | Key 1 | | E(K, $0^{128}$) |
| 6 (0x34128) | | | |
| 7 (0x34130) | Key 2 | len(AAD)$^T$ | |
| 8 (0x34138) | | | |
| 9 (0x34140) | Key 3 | H | |
| 10 (0x34148) | | | |

Table 10-30. AESU Context Registers for Integrity Modes (continued)

| Context Register # (byte address) | Cipher Mode providing only Data Integrity | | |
|---|---|---|---|
| | XCBC-MAC | GCM-GHASH | CMAC (OMAC1) |
| 11 (0x34150) | | len(AAD)$^{C}$ | |

**Notes:**
Context register 12 is unused for these modes
* Used only in ICV mode—must be written at start of new message for ICV checking
$^{C}$ Length of data processed with current descriptor (in bits)
$^{T}$ Length of total data (in bits)

## Context and Operation for XCBC-MAC Cipher Mode

XCBC-MAC cipher mode is an authentication-only mode of AES. Normal CBC-MAC runs AES in CBC cipher mode and assigns the final ciphertext result as the MAC. XCBC-MAC supports only 16-byte keys and extends the normal CBC-MAC as follows:

1. 3 keys are precomputed
   a. K1 = AES-Encrypt (K, $\{0x01\}^{16}$). [1]
   b. K2 = AES-Encrypt (K, $\{0x02\}^{16}$).
   c. K3 = AES-Encrypt (K, $\{0x03\}^{16}$).
2. Compute $C_{n-1}$ = AES-CBC ($P_1$, 0, K1) ... AES-CBC ($P_{n-1}$, $C_{n-2}$, K1)
3. If $|P_n|$ = block size (128 bits)
   then: MAC = AES-CBC ( $P_n \oplus K2$ , $C_{n-1}$, K1)
   else: MAC = AES-CBC ( $(P_n || 10^i) \oplus K3$ , $C_{n-1}$, K1)

In XCBC-MAC cipher mode, AUX0=1 means that the final data block is not XORed with K2 or K3, so that message processing can be interrupted and later continued after a context switch. AUX1=1 disables computation of keys K1, K2, and K3, and instead expects these keys to be placed in key registers 5–6 (K1), context registers 7–8 (K2) and 9–10 (K3). If AUX1=0, computed keys are placed into context registers 5–10. AUX2=1 enables XCBC-MAC with ICV. In XCBC-MAC with ICV, the received MAC is supplied in context registers 3–4 and compared to the computed MAC in context registers 1–2.

Operation of the AESU in XCBC-MAC cipher mode requires the following steps (note these steps are performed automatically in channel-driven access):

1. Reset
2. Program the mode register as follows:
   a. Set the cipher mode to XCBC-MAC (encode/decode bit is ignored)
   b. Set AUX0 = 1 if processing of the message is going to be interrupted and later continued after a context switch. Set AUX0 = 0 if this is the last (or only) part of the message so that the final MAC can be generated.

[1] Notation: $\{01\}^{16}$ means the byte 0x01 repeated 16 times

    c. Set AUX1 = 1 if keys K1, K2 and K3 are loaded to key registers 5–6, context registers 7–8 and 9–10, respectively. Otherwise, set AUX1=0, and put K into key registers 1–2, so that keys K1, K2, and K3 can be computed and written to context registers 5–6, 7–8, and 9–10, respectively.

    d. Set AUX2 = 1 if using XCBC-MAC with ICV.

3. Load Key (K if AUX1=0; K1 if AUX1=1)
4. Load Context
5. Set Key size
6. Set data size
7. While available:
    a. Load data blocks
8. Write to the end of message register
9. Read MAC from context registers 1-2
10. For XCBC-MAC with ICV, check ICCR bits in the status register

#### Context and Operation for GCM-GHASH Cipher Mode

GCM-GHASH denotes the authentication part of GCM cipher mode, and is described in Section 10.7.1.11.3, “Context for Confidentiality and Data Integrity Cipher Modes”.

#### Context and Operation for CMAC (OMAC1) Cipher Mode

CMAC cipher mode is an authentication-only mode of AES. CMAC may be specified using the following notation:

- E(K,L) denotes the AES-encrypt function;
- xtime(L) is defined as follows, where L is a 128-bit vector with L[127] as most significant bit:
  — If L[127]=0, then xtime(L)=L<<1 (where ‘<<‘ denotes bitwise left shift)
  — Else xtime(L) = (L<<1) XOR 0x87.

Using this notation, the specification of CMAC is as follows:

1. Two keys are precomputed as follows:
    a. K1 = xtime(E (K, $\{0\}^{128}$)).[1]
    b. K2 = xtime(K1).
2. Compute $C_{n-1}$ = AES-CBC ($P_1$, 0, K) ... AES-CBC ($P_{n-1}$, $C_{n-2}$, K)
3. If $|P_n|$ = block size (128 bits)
   then: MAC = AES-CBC ( $P_n \oplus K1$ , $C_{n-1}$, K)
   else: MAC = AES-CBC ( $(P_n||10^i) \oplus K2$ , $C_{n-1}$, K)

[1] Notation: $\{0\}^{128}$ means the bit 0 repeated 128 times

In CMAC cipher mode the received MAC is placed in context registers 3–4, and the computed MAC is put in registers 1–2 if AUX0=1. Context registers 5-6 are used to provide $E(K, \{0\}^{128})$ if AUX1=1, so that K1 and K2 can be computed after context switch without a time penalty. The computed value of $E(K, \{0\}^{128})$ is always stored in context registers 5-6 to be available for saving context in case of context switching.

Operation of the AESU in CMAC cipher mode requires the following steps (note these steps are performed automatically in channel-driven access):

1. Reset
2. Program the AESU mode register as follows:
   a. Set cipher mode to CMAC (encode/decode bit is ignored)
   b. Set AUX0 = 1 if processing of the message is going to be interrupted and later continued after a context switch. Set AUX0 = 0 if this is the last (or only) part of the message so that the final MAC can be generated.
   c. Set AUX1 = 1 for keys K1 and K2 to be derived from $E(K, \{0\}^{128})$ that is loaded into context registers 5–6. Otherwise, set AUX1=0, and CMAC computes $E(K, \{0\}^{128})$.
   d. Set AUX2 = 1 if using CMAC with ICV.
3. Load key
4. Load context
5. Set key size
6. Set ICV size for computed/received MAC (8, 10, 12, 14 or 16 bytes, default is 16)—ignored if AUX0 = 1
7. Set data size
8. While available:
   a. Load data blocks
9. Write to the end of message register
10. Read MAC from context registers 1-2
11. For CMAC with ICV, check ICCR bits in the status register

#### 10.7.1.11.3 Context for Confidentiality and Data Integrity Cipher Modes

The context registers for the different cipher modes which provide both confidentiality and data integrity are summarized in Table 10-31. The registers are described in more detail in the following subsections.

**Table 10-31. AESU Context Registers for Modes Providing Confidentiality and Integrity**

| Context Register #<br>(byte address) | Cipher Mode providing Confidentiality and Integrity | |
|---|---|---|
| | CCM | GCM |
| 1 (0x34100) | IV* / MAC | Computed MAC |
| 2 (0x34108) | | |

Table 10-31. AESU Context Registers for Modes Providing Confidentiality and Integrity (continued)

| Context Register #<br>(byte address) | Cipher Mode providing Confidentiality and Integrity | |
|---|---|---|
| | CCM | GCM |
| 3 (0x34110) | Encrypted MAC** /<br>Decrypted MAC /<br>Encrypted Counter | Received MAC** |
| 4 (0x34118) | | |
| 5 (0x34120) | Counter* | Counter |
| 6 (0x34128) | | |
| 7 (0x34130) | Counter Modulus Exponent*<br>(header size/ MAC size)*** | len(AAD)$^T$ |
| 8 (0x34138) | — | len(IV)$^T$ |
| 9 (0x34140) | — | $Y_0$ |
| 10 (0x34148) | | |
| 11 (0x34150) | — | len(AAD)$^C$ |
| 12 (0x34158) | | len(IV)$^C$ |

* Must be written at start of new message, except if zero

** Needed only in ICV mode—must be written at start of new CCM decryption

*** The Header and MAC Sizes are internally constructed by the AES engine; then, that information is included inside context register 7 for context switching purposes

$^C$ length of data processed with current descriptor (in bits)

$^T$ length of total data (in bits)

— don’t care

## Context for CCM Cipher Mode

The SEC AESU is capable of performing single pass encryption and MAC generation. The host is required to order the CCM context in such a way that the context can be fetched as a contiguous string into the context registers, prior to encryption/MAC generation or decryption/MAC validation. The context register contents for CCM cipher mode is summarized in Figure 10-30 and further described below.

### NOTE

AES-CCM mode does not support zero-length AAD and zero-length payload simultaneously. Either the AAD length or the payload length must be at least 1 byte.

| | | Context Registers | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Encrypt (outbound) | Inputs | IV | | 0 | | Initial Counter | | Counter Modulus Exponent |
| | Outputs | MAC | 0 | MIC | 0 | | | |
| Decrypt (inbound) | Inputs | IV | | MIC | 0 | Initial Counter Value | | Counter Modulus Exponent |
| | Outputs | Computed MAC | 0 | Decrypted MAC | 0 | | | |

**Figure 10-30. AESU CCM Context Registers**

### Context and Operation for CCM Encryption/MAC Generation

The context for CCM encryption/MAC generation (shown in Figure 10-30) is as follows:

- Registers 1–2 contain the session-specific 128-bit initialization vector (from memory)
- Registers 3–4 contain 128 bits of zero padding
- Registers 5–6 contain the session specific initial counter value (from memory)
- Register 7 contains the counter modulus exponent.

  **Several current standards require a counter modulus exponent of 128 for CCM cipher mode. However, in order to support possible new standards the counter modulus exponent in AESU is a programmable field, which must be generated and stored along with other session-specific information for loading into the AESU context register prior to CCM encryption.**

Using the session-specific key and context described above, operation of the AESU for CCM encryption/MAC generation requires the following steps (note these steps are performed automatically in channel-driven access):

1. Initialize the IV, and encrypt with the symmetric key.
2. In CBC fashion, take the output of step 1, hash with the first block of plaintext, and encrypt with the symmetric key.
3. Continue as in step 2 until the final block of plaintext has been processed. The result of the encryption of the final block of plaintext with the symmetric key is the MAC tag. The full 128 bits of MAC data is written to context registers 1–2, for use in the next phase of CCM processing. Once the MAC tag has been generated (step 3), the MAC tag along with the plaintext is encrypted with the AESU operating in counter cipher mode.
4. The first item to be encrypted in counter cipher mode is the counter (initial counter value) from context registers 5–6. The counter is encrypted with the symmetric key, and the result is hashed with the MAC tag (retrieved from context registers 1–2) to produce the MIC (encrypted MAC), which is then stored in context registers 3–4. At the completion of CCM encrypt processing, this MIC is output to memory (per the descriptor pointer) for the host to append to the 802.11i frame.

**The AESU writes the full 128-bit MIC out to memory. The host must only append the most significant 64 bits to the frame as the MIC.**

5. The counter value is incremented, then encrypted with the symmetric key. The result is hashed with the first block of plaintext to produce the first block of cipher text. The ciphertext is placed in the AESU output FIFO.
6. The counter continues to be incremented, and encrypted with the symmetric key, with the result hashed with each successive block of plaintext, until all plaintext has been converted to ciphertext. The SEC controller manages FIFO reads and writes, fetching plaintext and writing ciphertext per the pointers provided in the descriptor. When all ciphertext and the MIC has been output, the CCM encrypt operation is complete.

### Context and Operation for CCM Decryption/MAC Regeneration

The context for CCM decryption/MAC regeneration (shown in Figure 10-30) is as follows:

- Registers 1–2 contain the session-specific 128-bit initialization vector (from memory)
- Registers 3–4 contain the MIC (from the received frame) plus 64 bits of zero padding
- Registers 5–6 contain the session-specific initial counter value (from memory)
- Register 7 contains the counter modulus exponent

  **Several current standards require a counter modulus exponent of 128 for CCM cipher mode. However, in order to support possible new standards the counter modulus exponent in AESU is a programmable field, which must be generated and stored along with other session-specific information for loading into the AESU context register prior to CCM encryption.**

Using the session-specific key and context described above, operation of the AESU for CCM decryption and MAC regenerationrequires the following steps (note these steps are performed automatically in channel-driven access):

1. Initialize the IV, and encrypt with the symmetric key. Simultaneously, the counter (Initial Counter Value) from Context Registers 5-6 is encrypted with the symmetric key. The result is hashed with the encrypted MAC (from Context Register 3-4), and the resulting original MAC is written to Context Reg 3-4, overwriting the encrypted MAC.

   **Strictly speaking, the counter is encrypted with the symmetric key: however the AESU should be set for “decrypt” to perform the counter and CBC processes in the correct order.**
2. The 802.11 frame header is hashed with the encrypted IV. (The AESU automatically determines the header length.) Simultaneously, the counter is incremented, and is then encrypted with the symmetric key. The result is then hashed with the first block of ciphertext to produce the first block of plaintext. The plaintext is placed in the AESU output FIFO, while simultaneously, in CBC fashion, a copy of the first block of plaintext is hashed with the output of encryption of the 802.11 frame header. The output is encrypted with the symmetric key.
3. As each ciphertext block is converted to plaintext, the plaintext is CBC encrypted. When the final plaintext block has been processed, the CBC MAC (MAC tag) is written to context registers 1–2. The first 64 bits of the MAC tag are compared to the MAC tag recovered in step 1.

### NOTE

> For both encrypt and decrypt operations, if the 802.11 frame is being processed as a whole (not split across multiple descriptors), the “Initialize” (AUX1) and “Final MAC” (AUX2) bits should be set in the AESU mode register.

### Options and Operation for GCM Cipher Mode

Galois counter mode (GCM) uses AES counter mode to achieve data confidentiality. Authentication is achieved by computing a GHASH message authentication code (GMAC) through performing repetitive multiplication-accumulate functions in a Galois field.

Normally, the initialization vector (which is provided through the input FIFO) is 96 bits. If it is 96 bits, then the initialization vector (IV) is padded with the value ($\{0\}^{31}1$)[1]; otherwise the IV is hashed using the GHASH (H, {}, IV) function, where H represents E($\{0\}^{128}$, K), E stands for encryption operation, and K represents the key used. The resulting value $Y_0$ (the padded IV or the GHASHed IV) is provided as the initial counter value to counter mode AES. The result of encrypting $Y_0$ is denoted E($Y_0$, K), and is used to generate the final MAC tag.

Data is encrypted or decrypted by XORing input data with the pseudorandom key stream generated by counter mode AES, starting with the *second* pseudorandom key block. The initial counter value $Y_0$ is incremented modulo $2^{32}$.

GCM cipher mode can optionally be used to perform only the authentication part (GHASH (H, AAD, ciphertext), where ‘AAD’ denotes ‘additional authenticated data’): this special sub-mode is called GCM-GHASH in this document. GCM-GHASH is implemented by setting AUX0 and specifying the appropriate encryption operation. The format of the context registers for GCM-GHASH mode is shown in Table 10-36.

GCM cipher mode also has option of automatically verifying that the received and computed MAC tags are identical. This cipher mode is called GCM with ICV and can be specified by setting AESU mode register bits 56, 57 and 62 to 1, and bit 61 to 0. GCM with ICV context format is shown in Table 10-35.

Messages (IV+AAD+text data) are fed in through the input FIFO, and are always processed in the following order: IV, AAD, text data, followed by the final MAC computation (where “text data” refers to plaintext or ciphertext to be operated on). The whole message, however, does not have to be processed in one GCM execution. It can be split and processed with multiple descriptors in multiple GCM runs separated by resets of the AESU block. The boundaries can be set at the end of any full block (16 bytes) of the stream IV+AAD+text data. Hence, any of the individual components (IV, AAD, or text data) can be split into multiple descriptors. Refer to Table 10-32 for proper AUX mode specification in this case and to Table 10-33 through Table 10-36 for proper context formatting under the different GCM options (encrypt, decrypt, GCM with ICV, or GCM-GHASH). It should be noted that in case of a late arrival of the MAC tag on the receiving side, the final MAC can be computed and verified against the received MAC in a separate descriptor after the rest of the message (IV+AAD+text data) has already been processed.

[1] Notation: $\{0\}^{31}1$ is defined to mean a string of thirty-one bits of 0 followed by a single bit of 1.

Operation of the AESU in GCM cipher mode requires the following steps (note these steps are performed automatically in channel-driven access):

1. Reset.
2. Set cipher mode to GCM or GCM with ICV and specify encrypt, decrypt in the AESU mode register. To perform GCM-GHASH (only GHASH (H, AAD, ciphertext) is computed) set AUX0 and specify encrypt. Set AUX2 and AUX1 bits according to Table 10-32.
3. Load key
4. Load (restore) context as needed (see Table 10-33 to Table 10-36).
5. Set key size
6. Set the size of the computed/received MAC (8, 12 or 16 bytes, default is 16)
7. Set data size
8. While available:
   a. Load IV into the input FIFO (1 or multiple blocks up to $2^{64}$ bits in total)
   b. Load AAD into the input FIFO (0 or multiple blocks up to $2^{64}$ bits in total)
   c. Load plaintext (for encryption) or ciphertext (for decryption) blocks into the input FIFO
   d. Unload ciphertext (for encryption) or plaintext (for decryption) blocks from the output FIFO
9. Write to the end of message register
10. Unload final ciphertext (for encryption) or plaintext (for decryption) blocks
11. Read (Save) context registers if another segment of the message is processed later
12. Read final GCM MAC from context registers 1-2, if AUX2 bit was set in mode register
13. For GCM with ICV, check ICCR bits in the AESU status register

### AESU Mode Register Auxiliary Bit Settings for GCM Cipher Modes

Table 10-32 shows the significance of the AUX bits (bits 58–60) in the AESU mode register, under different operating conditions.

**Table 10-32. GCM Cipher Mode Auxiliary Bit Definitions**

| Auxiliary Bit | Definitions | |
|---|---|---|
| | 0 | 1 |
| **AUX2 (bit 58)** | Do not compute MAC | Compute MAC |
| **AUX1 (bit 59)** | One of the following cases:<br>Descriptor contains the whole message (IV+AAD+text data)<br>Descriptor contains the whole IV and no or part of AAD or text data<br>Descriptor contains a non-final part of IV, AAD, text data (IV, AAD or text data split between descriptors)<br>Descriptor contains the final part of AAD or text data but no MAC is computed | One of the following cases:<br>Descriptor contains the final part of IV (IV split between descriptors)—len(IV)$^T$ needed<br>Descriptor contains the final part of text data and the final MAC is computed (AUX2=1) (text data split between descriptors)—len(AAD)$^T$, len(text data)$^T$ needed<br>Descriptor contains the whole text data but no or part of AAD and the final MAC is computed—len(AAD)$^T$, len(text data)$^T$ needed<br>Descriptor contains the final part of AAD and the final MAC is computed—len(AAD)$^T$, len(text data)$^T$ needed<br>Descriptor computes only MAC (based on restored context) but does not contain either IV, AAD or text data —len(AAD)$^T$, len(text data)$^T$ needed |
| **AUX0 (bit 60) and Encrypt** | -- | GHASH-only mode |
| **AUX0 (bit 60) and Decrypt** | The key is to be unrolled | The key is already unrolled |

AUX0 has different use depending on whether encryption or decryption is specified. For decryption, it determines whether the provided key should be first unrolled before processing starts, while in case of encryption it should generally be set to 0 unless GCM-GHASH cipher mode is desired. AUX2 determines whether the final MAC tag is to be computed or not. If AUX2 is set to 1, E(K, $Y_0$) and the last iteration of the GHASH(H, AAD, ciphertext) is going to be performed and then XORed to give the MAC tag. Hence, if the message is split into multiple descriptors, only the last one should have AUX2=1 for proper MAC tag computation. AUX1 is used to resolve the issues related to the splitting of messages into multiple descriptors. Table 10-32 shows the proper settings of AUX1 for several scenarios of message splitting. In general, whenever the final GHASH iteration needs to be computed (either for GHASH(H, {}, IV) or GHASH(H, AAD, ciphertext)), and the current length is not equal to total length for either IV, AAD, or text data, then AUX1 should be set to 1. Consequently, an AUX1 value of 1 also indicates that the context registers 9-10 need to provide the total length of IV, AAD, or text data for this to be accomplished.

### Context for GCM Cipher Modes

Table 10-33 to Table 10-36 describe the proper usage of context registers in case of encryption, decryption, GCM with ICV, and GCM-GHASH cipher mode settings, respectively. The context is in each case described in terms of the input context required for starting new GCM processing or continuation of processing after context switch, and in terms of the results (output) stored in context registers after GCM execution run is completed. The tables are followed by verbal descriptions of the different registers for the different options.

**Table 10-33. GCM Encryption Context**

| Context Register | GCM Encrypt (Outbound) | | | | |
|---|---|---|---|---|---|
| | Mode Register (ECM = 10, AUX0 = 0, CM = 01, ED = 1) | | | | |
| | AUX1 Value | | | AUX2 Value | |
| | Inputs | | | Outputs | |
| | | AUX1 = 1 | | | |
| | AUX1 = 0 | last AAD or text data segment, or MAC only | last IV segment | AUX2 = 0 | AUX2 = 1 |
| **1** | MAC<br>(Computed) | | | — | MAC<br>(Computed) |
| **2** | | | | | |
| **3** | — | | | — | |
| **4** | | | | | |
| **5** | $Y_i$<br>(Counter) | | | — | |
| **6** | | | | | |
| **7** | — | len(AAD)$^{T*}$ | — | — | |
| **8** | — | len(text data)$^{T*}$ | len(IV)$^{T}$ | — | |
| **9** | $Y_0$<br>(Initial Counter) | | | — | |
| **10** | | | | | |
| **11** | len(AAD)$^{C*}$ | | | — | |
| **12** | len(IV)$^{C*}$ | | | — | |

* Must be written at the start of a new message, except if zero

C length of data processed with current descriptor (in bits)

T length of total data (in bits)

-- don’t care

## Table 10-34. GCM Decryption Context

| Context Register | GCM Decrypt (Inbound) | | | | |
|---|---|---|---|---|---|
| | Mode Register (ECM = 10, AUX0 = 0 or 1, CM = 01, ED = 0) | | | | |
| | AUX1 Value | | | AUX2 Value | |
| | Inputs | | | Outputs | |
| | AUX1 = 0 | AUX1 = 1 | | AUX2 = 0 | AUX2 = 1 |
| | | last AAD or text data segment, or MAC only | last IV segment | | |
| 1 | MAC<br>(Computed) | | | — | |
| 2 | | | | | |
| 3 | — | | | — | MAC<br>(Computed) |
| 4 | | | | | |
| 5 | Yi<br>(Counter) | | | — | |
| 6 | | | | | |
| 7 | — | len(AAD)$^{T*}$ | — | — | |
| 8 | — | len(text data)$^{T*}$ | len(IV)$^{T*}$ | — | |
| 9 | $Y_0$<br>(Initial Counter) | | | — | |
| 10 | | | | — | |
| 11 | len(AAD)$^{C*}$ | | | — | |
| 12 | len(IV)$^{C*}$ | | | | |

* Must be written at the start of a new message, except if zero

$^{C}$ length of data processed with current descriptor (in bits)

$^{T}$ length of total data (in bits)

-- don't care

### Table 10-35. GCM with ICV Context

| Context Register | GCM with ICV (Inbound) | | | | |
|---|---|---|---|---|---|
| | Mode Register (ECM = 11, AUX0 = 0 or 1, CM = 01, ED = 0) | | | | |
| | AUX1 Value | | | AUX2 Value | |
| | Inputs | | | Outputs | |
| | | AUX1 = 1 | | | |
| | AUX1 = 0 | last AAD or text data segment, or MAC only | last IV segment | AUX2 = 0 | AUX2 = 1 |
| **1** | MAC<br>(Computed) | | | — | MAC<br>(Computed and truncated to icv_size most significant bytes) |
| **2** | | | | | |
| **3** | MAC<br>(Received) | | | — | |
| **4** | | | | | |
| **5** | $Y_i$<br>(Counter) | | | — | |
| **6** | | | | | |
| **7** | — | len(AAD)$^{T*}$ | — | — | |
| **8** | — | len(text data)$^{T*}$ | len(IV)$^{T}$ | — | |
| **9** | $Y_0$<br>(Initial Counter) | | | — | |
| **10** | | | | — | |
| **11** | len(AAD)$^{C*}$ | | | — | |
| **12** | len(IV)$^{C*}$ | | | | |

* Must be written at the start of a new message, except if zero

**C** length of data processed with current descriptor (in bits)

**T** length of total data (in bits)

-- don’t care

### Table 10-36. GCM-GHASH Context

| Context Register | GCM-GHASH (Only GHASH Computed) | | | | |
|---|---|---|---|---|---|
| | Mode Register (ECM = 10, AUX0 = 10, CM = 01, ED = 1) | | | | |
| | AUX1 Value | | | AUX2 Value | |
| | Inputs | | | Outputs | |
| | AUX1 = 0 | AUX1 = 1 | | AUX2 = 0 | AUX2 = 1 |
| | | last AAD or text data segment, or MAC only | last IV segment | | |
| 1 | MAC (Computed) | | | — | MAC (Computed) |
| 2 | | | | | |
| 3 | — | | | — | |
| 4 | | | | | |
| 5 | — | | | — | |
| 6 | | | | | |
| 7 | — | len(AAD)$^{T*}$ | — | — | |
| 8 | — | len(text data)$^{T*}$ | len(IV)$^{T}$ | — | |
| 9 | H$^{*}$ | | | — | |
| 10 | | | | | |
| 11 | len(AAD)$^{C}$ | | | — | |
| 12 | — | | | — | |

* Must be written at the start of a new message, except if zero

$^{C}$ length of data processed with current descriptor (in bits)

$^{T}$ length of total data (in bits)

-- don't care

The context registers may be described as follows:

- Registers 1–2 contain the intermediate MAC value. This needs to be provided only when switching context during additional authenticated data (AAD) and/or text data processing (AAD+text data stream split into multiple descriptors).

  On the output side, these registers contain either the intermediate MAC tag in case of context switching (requires AUX2=0) or the final MAC tag at the end of processing (if AUX2=1). If AUX2=0 on the last descriptor processing a particular message, then these registers contain the partially computed GHASH(H, AAD, ciphertext), where the last GHASH iteration is not computed.

In the case of GCM with ICV, the final MAC tag written here as the result of GCM processing is truncated to 8, 12, or 16 (no truncation) bytes as defined in ICV size register. Note that any size from 1 to 16 bytes can be specified in ICV size register but any value other than 8 or 12 automatically defaults to 16 bytes.

- Registers 3–4 contain the received MAC tag, in case of inbound processing using GCM with ICV. This can be a 8, 12 or 16-byte block as specified by the ICV size register.
- Registers 5–6 contain the counter value $Y_i$, which is required only if restoring the context to continue processing a message. Note that the same value read when saving context should be written to these registers when restoring the context, since it is automatically incremented after every processed block.

  In the case of GCM-GHASH, these registers are not used.
- Register 7 contains the total length of the additional authenticated data (AAD) in bits. This is the total AAD length irrespective of whether AAD is split in multiple descriptors. It is required when AUX1=1 and the current descriptor processes the last segment of AAD or text data. It is also required if the whole message is already processed and the current descriptor only computes the final MAC tag.
- Register 8 contains the total length of the plaintext/ciphertext or IV in bits. This is required only when AUX1=1 (see Table 10-32). If the current descriptor processes the last segment of the IV, then total IV length should be provided; otherwise, the total length of text data should be provided.
- Registers 9–10 contain the initial counter value $Y_0$. Normally, this value is a result of the IV stream processing and needs to be provided only if the message is split into multiple descriptors and for those descriptors that come after IV processing is complete. Otherwise, the value provided here is ignored and overwritten with computed $Y_0$.

  In case of GCM-GHASH cipher mode setting, the constant H from GHASH(H, AAD, ciphertext) should be provided in these registers. Note that in the general case this may not be to be equal to $E(K, \{0\}^{128})$ where K is a key as defined for GCM.
- Register 11 contains the length (in bits) of the AAD part processed in the current descriptor. If the current descriptor does not process AAD, then the register should be zero. If AAD is not split into multiple descriptors, then this field should contain the total AAD length. The value written here should be divisible by 128 for all AAD segments except for the last one, which can be any number of bits. Note, however, that the actual AAD stream supplied to the AES engine through the FIFOs has to be zero-padded to an integral number of 16-byte blocks.
- Register 12 contains the length (in bits) of the IV part processed in the current descriptor. Similar remarks apply for IV in register 12 as for AAD in register 11.

  In case of GCM-GHASH, this register is not used.

### Example of Context in GCM Encryption

For illustrative purposes we consider the case of a GCM encrypt operation that generates the final MAC tag, where the whole message is small enough to be processed with one descriptor. AESU mode register bits 56-63 (ECM, AUX, CM, and ED) should be set to 10_100_01_1. Only context registers 11–12 must be written with the bit lengths of AAD and IV, respectively. The bit length of text data should be written to the data size register, up to a size of $2^{19}$ bits. IV, AAD, and text data in that order should be sent through the input FIFO and the result (text data) read from the output FIFO as available. At the end, the final MAC tag is read from context registers 1–2.

### 10.7.1.12 AESU Key Registers

The format of the AESU key registers is shown in Figure 10-31. These registers may hold 16, 24, or 32 bytes of key data, with the first 8 bytes of key data written to key 1.Any key data written to bytes beyond the key size (as specified in the key size register) is ignored. The key data registers are cleared when the AESU is reset or re-initialized. If these registers are modified during message processing, a context error is generated.

The key registers may be read when changing context in decrypt mode. To resume processing, the value read must be written back to the key registers and the “restore decrypt key” bit must be set in the mode register. This eliminates the overhead of expanding the key prior to starting decryption when switching context.

| | 0 | 63 | |
|---|---|---|---|
| Field | Key 1U Register | | Key 1U |
| Reset | 0 | | |
| R/W | R/W | | |
| Addr | AESU 0x3_4400 | | |

| | | |
|---|---|---|
| Field | Key 1L Register | Key 1L |
| Reset | 0 | |
| R/W | R/W | |
| Addr | AESU 0x3_4408 | |

| | | |
|---|---|---|
| Field | Key 2U Register | Key 2U |
| Reset | 0 | |
| R/W | R/W | |
| Addr | AESU 0x3_4410 | |

| | | |
|---|---|---|
| Field | Key 2L Register | Key 2L |
| Reset | 0 | |
| R/W | R/W | |
| Addr | AESU 0x3_4418 | |

**Figure 10-31. AESU Key Registers**

#### 10.7.1.12.1 AESU FIFOs

AESU uses an input FIFO/output FIFO pair to hold data before and after the encryption process. Normally, the channels control all access to these FIFOs. For host-controlled operation, a write to anywhere in the AESU FIFO address space enqueues data to the AESU input FIFO, and a read from anywhere in the AESU FIFO address space dequeues data from the AESU output FIFO.

Writes to the input FIFO go first to a staging register which can be written by byte, word (4 bytes), or dword (8 bytes). When all 8 bytes of the staging register have been written, the entire dword is automatically enqueued into the FIFO. If any byte is written twice between enqueues, it causes an error interrupt of type AE from the EU. When writing the last portion of data, it is not necessary to write all 8 bytes. Any last bytes remaining in the staging register are automatically padded with zeros and forced into the input FIFO when the AESU end of message register is written.

The output FIFO is readable by byte, word, or dword. When all 8 bytes of the head dword have been read, that dword is automatically dequeued from the FIFO so that the next dword (if any) becomes available for reading. If any byte is read twice between dequeues, it causes an error interrupt of type AE from the EU.

Overflows and underflows caused by reading or writing the AESU FIFOs are reflected in the AESU interrupt status register.

The AESU fetches data 128 bits at a time from the input FIFO. During processing, the input data is encrypted or decrypted and the results are placed in the output FIFO. The output size is the same as the input size.

The input FIFO may be written any time the number of dwords currently in the input FIFO (as indicated by the IFL field of the AESU status register) is less than 32. There is no limit on the total number of bytes in a message. The number of bits in the final message block must be set in the data size register.

The output FIFO may be read any time the OFR signal is asserted (as indicated in the AESU status register). This indicates that the number of bytes in the output FIFO is at or above the threshold specified in the mode register.

## 10.7.2 ARC4 Execution Unit (AFEU)

This section contains details about the ARC4 execution unit (AFEU), including modes of operation, status and control registers, S-box memory, and FIFOs.

Most of the registers described here would not normally be accessed by the host. They are documented here mainly for debug purposes. In typical operation, the AFEU is used through channel-controlled access, which means that most reads and writes of AFEU registers are directed by the SEC channels. Driver software would perform host-controlled register accesses only on a few registers for initial configuration and error handling.

#### 10.7.2.1 AFEU Mode Register

As shown in Figure 10-32, the AFEU mode register contains three bits which are used to program the AFEU. The mode register is cleared when the AFEU is reset or re-initialized. Setting a reserved mode bit generates a data error. If the mode register is modified during processing, a context error is generated.

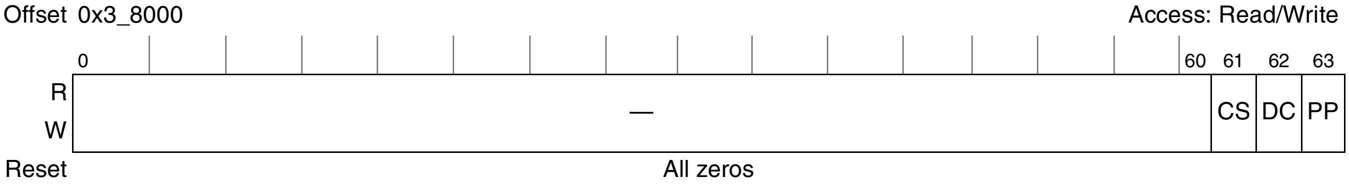


**Figure 10-32. AFEU Mode Register**

Table 10-37 describes AFEU Mode Register fields.

**Table 10-37. AFEU Mode Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| The following bits are described for information only. They are not under direct user control. | | |
| 0–55 | — | Reserved |
| The following bits are controlled through the MODE0 field of the descriptor header. | | |
| 56–60 | — | Reserved |
| 61 | CS | Context Source. If set, this causes the context to be moved from the input FIFO into the S-box prior to starting encryption/decryption. Otherwise, context should be directly written to the context registers or context should be generated automatically through key permutation. Context source is only checked if the prevent permute bit is set.<br>0 Context not from FIFO (written directly to context register addresses)<br>1 Context from input FIFO |
| 62 | DC | Dump Context. If set, this causes the context to be moved from the S-box to the output FIFO following assertion AFEU’s done interrupt.<br>0 Do not dump context<br>1 After cipher, dump context |
| 63 | PP | Prevent Permute. Normally, AFEU receives a key and uses that information to randomize the S-box. If reusing a context from a previous descriptor, this bit should be set to prevent AFEU from re-performing this permutation step.<br>0 Perform S-box permutation<br>1 Do not permute |

#### 10.7.2.2 AFEU Key Size Register

As displayed in Figure 20-60, this value indicates the number of bytes of key memory that should be used in performing S-box permutation. Any key data beyond the number of bytes in the key size register is ignored. This register is cleared when the AFEU is reset or re-initialized. If the key size specified is less than 1 or greater than 16, a key size error is generated. If the key size register is modified during processing, a context error is generated. Note: Although the AFEU supports key lengths as short as 1 byte, a 1 byte key offers little security. Most applications of ARC4 specify keys of 5-16 bytes.

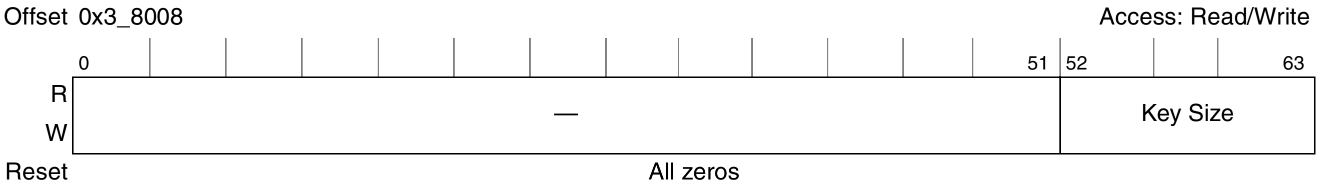


**Figure 10-33. AFEU Key Size Register**

> **NOTE**
>
> The device driver must create properly formatted descriptors for situations requiring a key permute prior to ciphering. When using host-controlled access (typically for debug), the user must set the AFEU mode register to perform ‘permute with key’, then write the key data to AFEU Key Registers, then write the key size to the key size register. The AFEU starts permuting the memory with the contents of the key registers immediately after the key size is written.

### 10.7.2.3 AFEU Context/Data Size Register

The AFEU context/data size register (shown in Figure 20-64), specifies the number of bits of context or data to be processed by the AFEU.

In channel-driven access, the necessary writes to this register are performed automatically, based on information contained in the descriptors.

In host-driven access, the correct order of operations for an AFEU operation with context loading is as follows:

1. Write the AFEU mode register, with 'Context Source' and 'Prevent Permute' set.
2. Write the 259 bytes of previously saved S-Box (256 bytes) and counters (3 bytes) to the AFEU input FIFO.
3. Write 2072 (bits) to the AFEU context/data size register
4. Begin writing the data to the AFEU Input FIFO. If the total data size is > 256 bytes, monitor the input FIFO level (IFL) in the AFEU Status Register to avoid overflowing the Input FIFO. Use the Output FIFO Level (OFL) to avoid underflowing the Output FIFO.
5. After writing the final data to the Input FIFO, write the data size (in bits) to the AFEU context/data size register. The data size written must be an integral number of bytes (bits 61:63 must be zero) or the AFEU will generate a data size error. The AFEU performs additional checking on bits 57:60 to determine the number of bytes of data from the final Input FIFO write to permute with the S-Box.

This register is cleared when the AFEU is reset or re-initialized, shown in Figure 10-34.

Offset 0x3_8010 Access: Read/Write

| | 0–51 | 52–63 |
|---|---|---|
| R<br>W | — | Data Size |
| Reset | All zeros | |

Figure 10-34. AFEU Context/Data Size Register

### 10.7.2.4 AFEU Reset Control Register

This register, as shown in Figure 10-35, allows 3 levels reset that effect the AFEU only, as defined by 3 self-clearing bits. It should be noted that the AFEU executes an internal reset sequence for hardware reset, SW_RESET, or module initialization, which performs proper initialization of the S-box. To determine when this is complete, observe the RESET_DONE bit in the AFEU status register.

Offset 0x3_8018 Access: Read/Write

| | 0–60 | 61 | 62 | 63 |
|---|---|---|---|---|
| R<br>W | — | RI | MI | SR |
| Reset | All zeros | | | |

Figure 10-35. AFEU Reset Control Register

Table 10-38 describes AFEU reset control register fields.

Table 10-38. AFEU Reset Control Register Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–60 | — | Reserved |
| 61 | RI | Reset Interrupt. Writing this bit active high causes AFEU interrupts signaling done and error to be reset. It further resets the state of the AFEU interrupt status register.<br>0 Do not reset<br>1 Reset interrupt logic |
| 62 | MI | Module initialization resets everything reset by SR, with the exception of the AFEU interrupt mask register.<br>0 Do not reset<br>1 Reset most of AFEU |
| 63 | SR | Software reset is functionally equivalent to hardware reset (the RESET# pin), but only for AFEU. All registers and internal state are returned to their defined reset state. On negation of SW_RESET, the AFEU enters a routine to perform proper initialization of the S-box.<br>0 Do not reset<br>1 Full AFEU reset |

### 10.7.2.5 AFEU Status Register

This status register, shown in Figure 10-36, reflect the state of AFEU internal signals.

The AFEU status register is read only. Writing to this location results in address error being reflected in the AFEU interrupt status register.

Offset 0x3_8028 Access: Read only

| | 0 39 | 40 47 | 48 55 | 56 57 | 58 | 59 60 | 61 | 62 | 63 |
|---|---|---|---|---|---|---|---|---|---|
| R | | OFL | IFL | — | HALT | — | EI | DI | RD |
| W | | | | | | | | | |
| Reset | All zeros | | | | | | | | |

**Figure 10-36. AFEU Status Register**

Table 10-39 describes AFEU Status Register fields.

**Table 10-39. AFEU Status Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–39 | — | Reserved |
| 40-47 | OFL | The number of dwords currently in the output FIFO |
| 48-55 | IFL | The number of dwords currently in the input FIFO |
| 56-57 | — | Reserved |
| 58 | HALT | Halt. Indicates that the AFEU has halted due to an error.<br>0 AFEU not halted<br>1 AFEU halted<br>**Note:** Because the error causing the AFEU to stop operating may be masked before reaching the interrupt status register, the AFEU interrupt status register is used to provide a second source of information regarding errors preventing normal operation. |
| 59–60 | — | Reserved |
| 61 | EI | Error interrupt: This status bit reflects the state of the error interrupt signal, as sampled by the controller interrupt status register (Section 10.5.4.2.2, “Interrupt Status Register (ISR)”).<br>0 AFEU is not signaling error<br>1 AFEU is signaling error |
| 62 | DI | Done interrupt: This status bit reflects the state of the done interrupt signal, as sampled by the controller interrupt status register (Section 10.5.4.2.2, “Interrupt Status Register (ISR)”).<br>0 AFEU is not signaling done<br>1 AFEU is signaling done |
| 63 | RD | Reset Done. This status bit, when high, indicates that AFEU has completed its reset sequence, as reflected in the signal sampled by the appropriate channel.<br>0 Reset in progress<br>1 Reset done<br>**Note:** Reset Done resets to 0, but has typically switched to 1 by the time a user checks the register, indicating the EU is ready for operation. |

### 10.7.2.6 AFEU Interrupt Status Register

The interrupt status register indicates which unmasked errors have occurred and have generated error interrupts to the channel. Each bit in this register can only be set if the corresponding bit of the AFEU interrupt mask register is zero (see Section 10.7.2.7, “AFEU Interrupt Mask Register”).

If the AFEU interrupt status register is non-zero, the AFEU halts and the AFEU error interrupt signal is asserted to the controller (see Section 10.5.4.2.2, “Interrupt Status Register (ISR)”). In addition, if the AFEU is being operated through channel-controlled access, then an interrupt signal is generated to the channel to which this EU is assigned. The EU error then appears in bit 55 of the channel status register (see Table 10-15) and generates a channel error interrupt to the controller.

If the interrupt status register is written from the host, 1s in the value written are recorded in the interrupt status register if the corresponding bit is unmasked in the interrupt mask register. All other bits are cleared. This register can also be cleared by setting the RI bit of the AFEU reset control register.

The definition of each bit in the AFEU interrupt status register is shown in Figure 10-37.

Offset 0x3_8030 Access: Read/Write

| | 0–50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | IE | ERE | CE | KSE | DSE | ME | AE | OFE | IFE | — | IFO | OFU | — |
| Reset | All zeros | | | | | | | | | | | | | |

**Figure 10-37. AFEU Interrupt Status Register**

Table 10-40 describes AFEU interrupt status register fields.

**Table 10-40. AFEU Interrupt Status Register Fields**

| Bits | Names | Description |
|---|---|---|
| 0-50 | — | Reserved |
| 51 | IE | Internal Error. An internal processing error was detected while performing encryption.<br>0 No error detected<br>1 Internal error |
| 52 | ERE | Early Read Error. The AFEU context memory or control was read while the AFEU was performing encryption.<br>0 No error detected<br>1 Early read error |
| 53 | CE | Context Error. The AFEU mode register, key register, key size register, data size register, or context memory is modified while AFEU processes data.<br>0 No error detected<br>1 Context error |
| 54 | KSE | Key Size Error. A value outside the bounds 1–16 bytes was written to the AFEU key size register<br>0 No error detected<br>1 Key size error |
| 55 | DSE | Data Size Error. A value that is not a multiple of 8 bits was written to the AFEU data size register:<br>0 No error detected<br>1 Data size error |
| 56 | ME | Mode Error. An illegal value was detected in the mode register. Note: writing to reserved bits in mode register is likely source of error.<br>0 No error detected<br>1 Mode error |

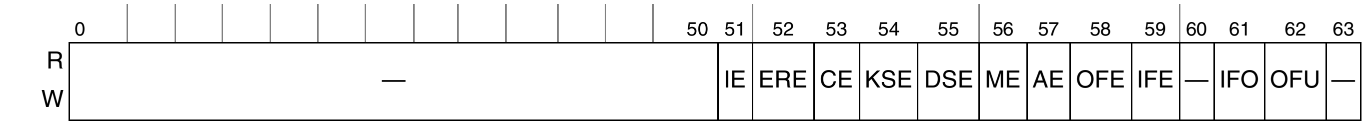
**Table 10-40. AFEU Interrupt Status Register Fields (continued)**

| Bits | Names | Description |
|---|---|---|
| 57 | AE | Address Error. An illegal read or write address was detected within the AFEU address space.<br>0 No error detected<br>1 Address error |
| 58 | OFE | Output FIFO Error. The AFEU output FIFO was detected non-empty upon write of AFEU data size register.<br>0 No error detected<br>1 Output FIFO non-empty error |
| 59 | IFE | Input FIFO Error. The AFEU Input FIFO was detected non-empty upon generation of done interrupt<br>0 Input FIFO non-empty error enabled<br>1 Input FIFO non-empty error disabled |
| 60 | — | Reserved |
| 61 | IFO | Input FIFO Overflow. The AFEU input FIFO was pushed while full.<br>1 Input FIFO has overflowed<br>0 No error detected<br>**Note:** When operated through channel-controlled access, the SEC implements flow control, which prevents input FIFO overflow—hence FIFO size is not a limit to data input in this case. When operated through host-controlled access, the AFEU cannot accept FIFO inputs larger than 256 bytes without overflowing. |
| 62 | OFU | Output FIFO Underflow. The AFEU output FIFO was read while empty.<br>0 No error detected<br>1 Output FIFO has underflow error |
| 63 | — | Reserved |

## 10.7.2.7 AFEU Interrupt Mask Register

The interrupt mask register, shown in Figure 10-38, controls the result of detected errors. For a given error (as defined in Section 10.7.2.6, “AFEU Interrupt Status Register”), if the corresponding bit in this register is set, the error is disabled; no error interrupt occurs and the interrupt status register is not updated to reflect the error. If the corresponding bit is not set, then upon detection of an error, the interrupt status register is updated to reflect the error, causing assertion of the error interrupt signal, and causing the module to halt processing.

Offset 0x3_8038 Access: Read/Write



| | 0–50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | IE | ERE | CE | KSE | DSE | ME | AE | OFE | IFE | — | IFO | OFU | — |

**Figure 10-38. AFEU Interrupt Mask Register**

Table 10-41 describes AFEU interrupt mask register fields.

Table 10-41. AFEU Interrupt Mask Register

| Bits | Names | Description |
|---|---|---|
| 0–50 | — | Reserved |
| 51 | IE | Internal Error. An internal processing error was detected while performing encryption.<br>0 Internal error enabled<br>1 Internal error disabled |
| 52 | ERE | Early Read Error. The AFEU Register was read while the AFEU was performing encryption.<br>0 Early read error enabled<br>1 Early read error disabled |
| 53 | CE | Context Error. An AFEU key register, the key size register, data size register, mode register, or context memory was modified while AFEU was performing encryption.<br>0 Context error enabled<br>1 Context error disabled |
| 54 | KSE | Key Size Error. A value outside the bounds 1–16 bytes was written to the AFEU key size register<br>0 Key size error enabled<br>1 Key size error disabled |
| 55 | DSE | Data Size Error. An inconsistent value was written to the AFEU data size register:<br>0 Data Size error enabled<br>1 Data size error disabled |
| 56 | ME | Mode Error. An illegal value was detected in the mode register.<br>0 Mode error enabled<br>1 Mode error disabled |
| 57 | AE | Address Error. An illegal read or write address was detected within the AFEU address space.<br>0 Address error enabled<br>1 Address error disabled |
| 58 | OFE | Output FIFO Error. The AFEU Output FIFO was detected non-empty upon write of AFEU data size register<br>0 Output FIFO non-empty error enabled<br>1 Output FIFO non-empty error disabled |
| 59 | IFE | Input FIFO Error. The AFEU Input FIFO was detected non-empty upon generation of done interrupt.<br>0 Input FIFO non-empty error enabled<br>1 Input FIFO non-empty error disabled |
| 60 | — | Reserved |
| 61 | IFO | Input FIFO Overflow. The AFEU Input FIFO was pushed while full.<br>0 Input FIFO overflow error enabled<br>1 Input FIFO overflow error disabled |
| 62 | OFU | Output FIFO Underflow. The AFEU Output FIFO was read while empty.<br>0 Output FIFO underflow error enabled<br>1 Output FIFO underflow error disabled |
| 63 | — | Reserved |

### 10.7.2.8 AFEU End of Message Register

The end of message register in the AFEU, displayed in Figure 10-39, is used to signal the AFEU that all data to be processed has been written to the input FIFO (in channel-driven access, this signaling is done automatically). Before this register is written, the AFEU does not process the last block of data in its input FIFO. Once the end of message register is written, the AESU processes any remaining data in the input FIFO and generates the done interrupt. If the DC(dump context) bit in the AFEU mode register is set, the context is written to the output FIFO following the last message word.

Any value written to the end of message register has the same effect. A read of the AFEU end of message register always returns a zero value.

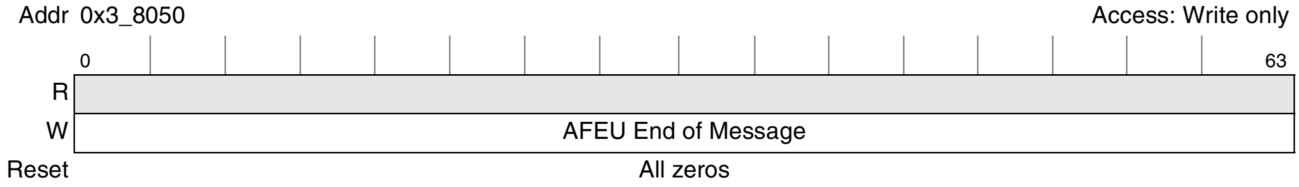


Figure 10-39. AFEU End of Message Register

### 10.7.2.9 AFEU Context

An ARC4 encryption session begins with an initial key permutation to generate the initial S-box state. After that each subsequent message in the same session makes use of the S-box state and also modifies the S-box state.

To implement this behavior using the AFEU, the first descriptor of a session must perform a key permute operation. If there are additional messages in the same session, then at the end of each descriptor execution the S-box state must be dumped out as context so that the next message of the session can re-load that context.

AFEU context consists of two parts:

- AFEU context memory — a 256-byte SRAM that holds the current S-box contents
- AFEU context memory pointer register — holds the internal context pointers that are updated with each byte of message processed. These pointers correspond to the values of I, J, and Sbox[I+1] in the ARC4 algorithm.

There is no standard data format for S-box state information. To ensure proper AFEU operation, the AFEU context should only be written with data that was read from the AFEU context during a previous operation.

#### 10.7.2.9.1 Writing AFEU Context

In the default mode of operation, the key and key size are provided to the AFEU. The initial memory values in the S-box are permuted with the key to create new S-box values, which are used to encrypt the plaintext.

If the "prevent permute" (PP) mode bit is set in the AFEU mode register (see Section 10.7.2.1, "AFEU Mode Register"), then the AFEU does not require a key, but instead requires context to set the S-box state before processing data. This mode is used to resume processing in an ARC4 session using a previously computed S-box. In this case, the steps in the processing are as follows:

1. Write the context to the AFEU through the context registers or through the input FIFO (as selected by the CS mode bit).
2. Write the context length to the context/data length register (see Section 10.7.2.3, "AFEU Context/Data Size Register").
3. Write the message data size to the context/data register.
4. Write the message data.

If the context registers are written during message processing or hen the PP bit is not set, a context error is generated.

For more information about writing AFEU context through the input FIFO, see Section 10.7.2.10.1, "AFEU FIFOs."

#### 10.7.2.9.2 Reading AFEU Context

Once message processing is complete and the output data has been read, the AFEU S-box state can be read out either through the context registers or through the output FIFO, as selected by the "dump context" (DC) mode bit (see Section 10.7.2.1, "AFEU Mode Register").

Valid context data can only be read after AFEU has completed processing (as indicated by the done interrupt, as reflected in the "DI" bit of the AFEU Status Register, Section 10.7.2.5, "AFEU Status Register"). Reading context data before the module is done generates an error interrupt.

For more information about reading AFEU context through the output FIFO, see Section 10.7.2.10.1, "AFEU FIFOs."

### 10.7.2.10 AFEU Key Registers

AFEU uses two write-only key registers to seed the initial permutation of the AFEU S-box, in conjunction with the AFEU key size register. Any key data beyond the key size (specified in the key size register) is ignored. AFEU permutes starting with the first byte of key register 0, and uses as many bytes from the two key registers as necessary to complete the permutation. Reading either of these memory locations generates an address error interrupt.

#### 10.7.2.10.1 AFEU FIFOs

AFEU uses an input FIFO/output FIFO pair to hold data before and after the ciphering process. Normally, the channels control all access to these FIFOs. For host-controlled operation, a write to anywhere in the AFEU FIFO address space enqueues data to the AFEU input FIFO, and a read from anywhere in the AFEU FIFO address space dequeues data from the AFEU output FIFO.

When context is written to the input FIFO (see Section 10.7.2.9.1, "Writing AFEU Context"), the first context write must be in the address range 3_8E00-3_8E07. Similarly, when context is read from the output FIFO (see Section 10.7.2.9.1, "Writing AFEU Context"), the first context read must be in the address range 3_8E00-3_8E07. This causes any incomplete data word remaining in the output FIFO to be cleared out so that the context can be read.

Writes to the input FIFO go first to a staging register which can be written by byte, word (4 bytes), or dword (8 bytes). When all 8 bytes of the staging register have been written, the entire dword is automatically enqueued into the FIFO. If any byte is written twice between enqueues, it causes an error interrupt of type AE from the EU. When writing the last portion of data, it is not necessary to write all 8 bytes. Any last bytes remaining in the staging register are automatically padded with zeros and forced into the input FIFO when the AFEU end of message register is written.

The output FIFO is readable by byte, word, or dword. When all 8 bytes of the head dword have been read, that dword is automatically dequeued from the FIFO so that the next dword (if any) becomes available for reading. If any byte is read twice between dequeues, it causes an error interrupt of type AE from the EU.

Overflows and underflows caused by reading or writing the AFEU FIFOs are reflected in the AFEU interrupt status register.

## 10.7.3 Cyclical Redundancy Check Unit (CRCU)

This section contains details about the cyclical redundancy check unit (CRCU), including modes of operation, status and control registers, and FIFO.

Most of the registers described here would not normally be accessed by the host. They are documented here mainly for debug purposes. In typical operation, the CRCU is used through channel-controlled access, which means that most reads and writes of CRCU registers are directed by the SEC channels. Driver software would perform host-controlled register accesses only on a few registers for initial configuration and error handling.

### 10.7.3.1 ICV Checking in CRCU

This EU includes an ICV checking feature, that is, it can verify a message/CRC pair by calculating a raw CRC and comparing it to the polynomial specific residue. The pass/fail result of this check can be returned to the host either by interrupt by a writeback of EU status fields into host memory, but not by both methods at once.

To signal the ICV checking result by status writeback, turn on either the IWSE bit or AWSE bit in the channel configuration register (see Section 10.4.4.1, “Channel Configuration Register (CCR)”), and mask the CICV fail (CICVF) bit in the interrupt mask register (see Section 10.7.3.9, “CRCU Interrupt Mask Register”). In this case the normal done signaling (by interrupt or writeback) is undisturbed.

To signal the ICV checking result by interrupt, unmask the CICVF bit in the interrupt mask register and turn off the IWSE and AWSE bits in the channel configuration register. If there is no CRC mismatch, then the normal done signaling (by interrupt or writeback) occurs. When there is an CRC mismatch, there is an error interrupt to the host, but no done interrupt or writeback.

### 10.7.3.2 CRCU Mode Register

The mode register (shown in Figure 10-40) is used to program the function of the CRCU and is generally the first register written. A context error is generated if this register is written after processing has begun.

Addr 0x3_F000 Access: Read/Write

| | 0–56 | 57 | 58 | 59 | 60 | 61 | 62–63 |
|---|---|---|---|---|---|---|---|
| R<br>W | — | CICV | — | DOC | DOS | DIS | ALG |
| Reset | All zeros | | | | | | |

**Figure 10-40. CRCU Mode Register**

Table 20-46 describes the CRCU mode register fields:

| Bits | Name | Description |
|---|---|---|
| The following bits are described for information only. They are not under direct user control. | | |
| 0–56 | — | Reserved |
| 57 | CICV | ICV Check. Selects whether a comparison between the computed CRC and the residue should be performed.<br>0 Disabled.<br>1 Enable ICV check. Compares a CRC calculated across the message and received ICV against the stored residue. The comparison is performed prior to the bit manipulations controlled by the DOS and DOC bits. |
| 58 | — | Reserved |
| 59 | DOC | Disable Output Complement. Normal processing includes complementing the CRC result.<br>0 Normal operation. CRC result is complemented..<br>1 Disable output complement. |
| 60 | DOS | Disable Output Swap. Normal processing includes a bit swap and byte swap of the CRC result.<br>0 Normal operation. CRC result is bit swapped and byte swapped.<br>1 Disable output swapping. |
| 61 | DIS | Disable Input Swap. Normal processing includes a bit swap and byte swap of the input data.<br>0 Normal operation. Input data is bit swapped and byte swapped.<br>1 Disable input swapping. |
| 62-63 | ALG | Algorithm. Selects the CRC algorithm mode for the CRCU.<br>00 IEEE 802 mode. The CRC32 algorithm will be performed using the polynomial 0x04C11DB7 and the residue 0xC704DD7B will be used for ICV checking.<br>01 iSCSI mode. The CRC32c algorithm will be performed using the polynomial 0x1EDC6F41 and the residue 0x1C2D19ED will be used for ICV checking.<br>10 Static custom mode. The CRC remainder will be computed using the Control Register bits 32-63 as the polynomial and bits 0-31 as the residue.<br>11 Dynamic custom mode. The CRC remainder will be computed using the Key Register bits 32-63 as the polynomial and bits 0-31 as the residue. |

### 10.7.3.3 CRCU Key Size Register

The key size register stores the number of valid bytes in the dynamic polynomial written to the key register (Section 10.7.3.13, “CRCU Key Register”). The reset value is zero, and an illegal key size error is generated if a value other than zero, four or eight is written to the register. It is not necessary to write to this register, as the polynomial size is clearly fixed by the algorithm. A context error is generated if this register is written after processing has begun.

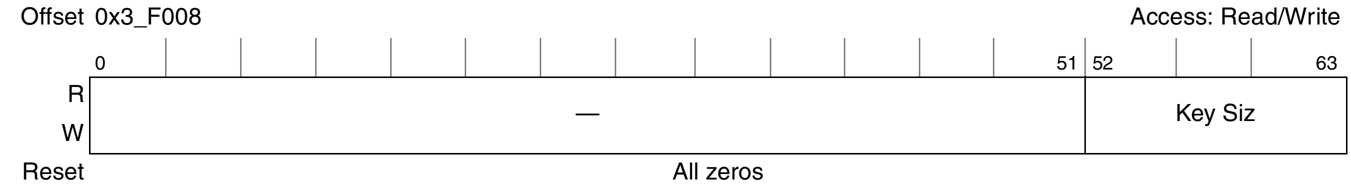


**Figure 10-41. CRCU Key Size Register**

### 10.7.3.4 CRCU Data Size Register

The data size register is written with the number of bits of data to be processed. Writing to this register puts the CRCU module into a busy state and starts data processing. This register may be written multiple times while data processing is in progress. The actual values written are ignored, although an error is generated if the value is not a multiple of 8 bits.

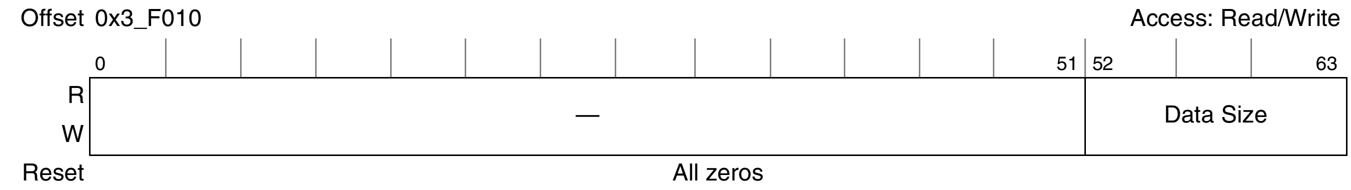


**Figure 10-42. CRCU Data Size Register**

### 10.7.3.5 CRCU Reset Control Register

The reset control register controls the reset/re-initialization of the block.

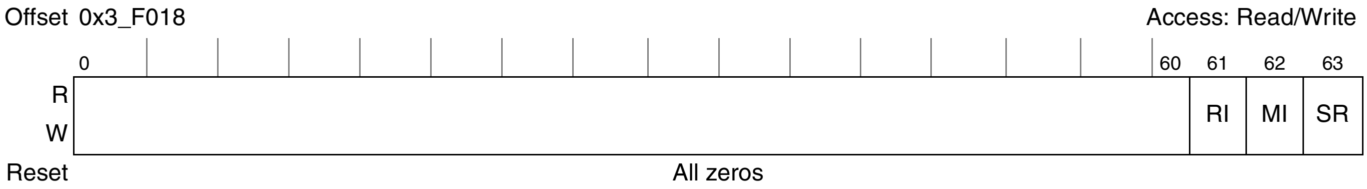


**Figure 10-43. CRCU Reset Control Register**

Table 10-42 describes CRCU reset control register fields.

**Table 10-42. CRCU Reset Control Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–60 | — | Reserved |
| 61 | RI | Reset Interrupt. Writing this bit active high causes CRCU interrupts signaling done and error to be reset. It further resets the state of the CRCU interrupt status register.<br>0 Do not reset<br>1 Reset interrupt logic |

Table 10-42. CRCU Reset Control Register Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 62 | MI | Module initialization resets everything reset by SR, with the exception of the CRCU interrupt mask register.<br>0 Do not reset<br>1 Reset most of CRCU |
| 63 | SR | Software reset is functionally equivalent to hardware reset (the RESET# pin), but only for CRCU. All registers and internal state are returned to their defined reset state. On negation of SW_RESET, the CRCU enters a routine to perform proper initialization of the S-box.<br>0 Do not reset<br>1 Full CRCU reset |

### 10.7.3.6 CRCU Control Register

The CRCU control register stores the coefficients of the residue and static polynomial used in custom CRC computations. Figure 10-44 shows the bit position of each coefficient.

Offset 0x020 Access: Read/Write

<table>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>23</th><th>24</th><th>25</th><th>26</th><th>27</th><th>28</th><th>29</th><th>30</th><th>31</th></tr>
<tr><td>R<br>W</td><td>$r^{31}$</td><td>$r^{30}$</td><td>$r^{29}$</td><td>$r^{28}$</td><td>$r^{27}$</td><td>$r^{26}$</td><td>$r^{25}$</td><td>$r^{24}$</td><td colspan="2">—</td><td>$r^{7}$</td><td>$r^{6}$</td><td>$r^{5}$</td><td>$r^{4}$</td><td>$r^{3}$</td><td>$r^{2}$</td><td>$r^{1}$</td><td>$r^{0}$</td></tr>
<tr><th></th><th>32</th><th>33</th><th>34</th><th>35</th><th>36</th><th>37</th><th>38</th><th>39</th><th>40</th><th>55</th><th>56</th><th>57</th><th>58</th><th>59</th><th>60</th><th>61</th><th>62</th><th>63</th></tr>
<tr><td>R<br>W</td><td>$g^{31}$</td><td>$g^{30}$</td><td>$g^{29}$</td><td>$g^{28}$</td><td>$g^{27}$</td><td>$g^{26}$</td><td>$g^{25}$</td><td>$g^{24}$</td><td colspan="2">—</td><td>$g^{7}$</td><td>$g^{6}$</td><td>$g^{5}$</td><td>$g^{4}$</td><td>$g^{3}$</td><td>$g^{2}$</td><td>$g^{1}$</td><td>$g^{0}$</td></tr>
</table>

Figure 10-44. CRCU Control Register

In Figure 20-84 $r^n$ (respectively $g^n$) represents the $n$'th residue (respectively polynomial) coefficient. The reset value of this register corresponds to the IEEE 802 CRC32 residue and polynomial coefficients. This register is static in that it is only reset by performing a software reset, and not by an EU reinitialization. This allows a platform-specific custom polynomial to be written to the register once and used many times. A context error is generated if this register is written after processing has begun. A polynomial error is generated if a value is written to this register which does not have a one in bit 0 (representing $g^0$).

### 10.7.3.7 CRCU Status Register

The CRCU status register provides general information on the status of the CRCU. A read of the status register captures a snapshot of CRCU's operating state at a particular moment in time. Of notable interest to the user are the three interrupt flags (error interrupt, done interrupt, and reset done). Writes to this register are ignored.

Offset 0x3_F028 Access: Read only

<table>
<tr><th></th><th>0</th><th>47</th><th>48</th><th>55</th><th>56</th><th>57</th><th>58</th><th>59</th><th>60</th><th>61</th><th>62</th><th>63</th></tr>
<tr><td>R</td><td colspan="2" rowspan="2"></td><td colspan="2">IFL</td><td colspan="2" rowspan="2">—</td><td>HALT</td><td colspan="2">ICCR</td><td>EI</td><td>DI</td><td>RD</td></tr>
<tr><td>W</td><td colspan="2"></td><td></td><td colspan="2"></td><td></td><td></td><td></td></tr>
<tr><td>Reset</td><td colspan="11">All zeros</td></tr>
</table>

Figure 10-45. CRCU Status Register

Table 10-43. CRCU Status Register Bit Definitions

| Bits | Field Name | Description |
|---|---|---|
| 0–47 | — | Reserved |
| 48–55 | IFL | Input FIFO Level: The number of dwords currently in the input FIFO |
| 56–57 | — | Reserved |
| 58 | HALT | Indicates when the CRCU core has halted due to an error.<br>0 CRCU not halted<br>1 CRCU core halted (Must be reset/re-initialized)<br>**Note:** Because the error causing the CRCU to stop operating may be masked to the interrupt status register, the status register is used to provide a second source of information regarding errors preventing normal operation. |
| 59–60 | ICCR | Integrity Check Comparison Result<br>00 No integrity check comparison was performed.<br>01 The integrity check comparison passed.<br>10 The integrity check comparison failed.<br>11 Reserved<br>**Note:** A passed or failed result is generated only if ICV checking is enabled and the algorithm selected is f9. |
| 61 | EI | Error interrupt.Reflects the state of the error interrupt signal, as sampled by the controller interrupt status register (Section 10.5.4.2.2, “Interrupt Status Register (ISR)”).<br>0 CRCU is not signaling error<br>1 CRCU is signaling error |
| 62 | DI | Done Interrupt: Reflects the state of the done interrupt signal, as sampled by the controller interrupt status register (Section 10.5.4.2.2, “Interrupt Status Register (ISR)”).<br>0 Processing not done<br>1 All bytes processed |
| 63 | RD | Reset Done: Indicates when the CRCU has completed its reset sequence, as reflected in the signal sampled by the appropriate channel.<br>0 Reset in progress<br>1 Reset done |

### 10.7.3.8 CRCU Interrupt Status Register

The CRCU interrupt status register indicates which unmasked errors have occurred and have generated error interrupts to the channel. Each bit in this register can only be set if the corresponding bit of the CRCU interrupt mask register is zero (see Section 10.7.3.9, “CRCU Interrupt Mask Register”). If a CRCU interrupt mask register bit is set, the corresponding CRCU interrupt status bit is always zero regardless of the error status.

If the CRCU interrupt status register is non-zero, then the CRCU halts and the CRCU error interrupt signal is asserted to the controller (see Section 10.5.4.2.2, “Interrupt Status Register (ISR)”). In addition, if the CRCU is being operated in channel-driven mode, then an interrupt signal is generated to the channel to which this EU is assigned. The EU error then appears in bit 55 of the channel status register (see Table 10-15) and generates a channel error interrupt to the controller.

Interrupt status register bits can be set by writes from the host, but only if the corresponding bit is cleared in the interrupt mask register. Bits masked by the interrupt mask register bits are always zero.

The CRCU interrupt status and interrupt mask registers can be cleared by programming the CRCU reset control register, as described in Section 10.7.3.5, “CRCU Reset Control Register.”

Offset 0x3_F030 Access: Read/Write

| | 0 … 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 59 60 | 61 | 62 63 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | ICE | PE | IE | ERE | CE | KSE | DSE | ME | AE | — | IFO | — |
| Reset | All zeros | | | | | | | | | | | | |

**Figure 10-46. CRCU Interrupt Status Register**

**Table 10-44. CRCU Interrupt Status Register Bit Definitions**

| Bits | Name | Description |
|---|---|---|
| 0–48 | — | Reserved |
| 49 | ICE | Integrity Check (CRC) Error:<br>0 No error detected<br>1 Integrity check error detected. An ICV (CRC) check was performed and the supplied ICV (CRC) did not match the one computed by the CRCU. |
| 50 | PE | Polynomial Error.<br>0 No error.<br>1 An invalid polynomial has been written to the Key or Control Register. |
| 51 | IE | Internal Error. Indicates the CRCU has been locked up and requires a reset before use.<br>0 No internal error detected<br>1 Internal error detected<br>**Note:** This bit is asserted any time an enabled error condition occurs and can only be cleared by resetting the CRCU. |
| 52 | ERE | Early Read Error. The CRCU context was read before the CRCU completed the requested operation.<br>0 No error detected<br>1 Early read error |
| 53 | CE | Context Error. The CRCU mode, key size, control, context, or key register was modified after processing had begun.<br>0 No error detected<br>1 Context error |
| 54 | KSE | Key Size Error. A value other than 0 or 4 has been written to the CRCU key size register.<br>0 No error detected<br>1 Key size error |
| 55 | DSE | Data Size Error. An non-byte-multiple size has been written to the CRCU data size register.<br>0 No error detected<br>1 Data size error |

Table 10-44. CRCU Interrupt Status Register Bit Definitions (continued)

| Bits | Name | Description |
|---|---|---|
| 56 | ME | Mode Error. Bit is set if any of these error conditions is detected:<br>• any reserved bit of the Mode register is set<br>• the ALG field of the Mode register contains an illegal value<br>0 No error detected<br>1 Mode error |
| 57 | AE | Address Error. An illegal read or write address was detected within the CRCU address space.<br>0 No error detected<br>1 Address Error |
| 58–60 | — | Reserved |
| 61 | IFO | Input FIFO Overflow. The CRCU Input FIFO was pushed while full.<br>0 No overflow detected<br>1 Input FIFO has overflowed<br>**Note:** When operated through channel-controlled access, the SEC implements flow control, and FIFO size is not a limit to data input size. When operated through host-controlled access, the CRCU cannot accept FIFO inputs larger than 256 bytes without overflowing. |
| 62–63 | — | Reserved |

### 10.7.3.9 CRCU Interrupt Mask Register

The CRCU interrupt mask register, controls the setting of bits in the CRCU interrupt status register, as described in Section 10.7.3.8, "CRCU Interrupt Status Register." If a CRCU interrupt mask register bit is set, then the corresponding interrupt status register bit is always zero.

Masking an error bit allows for a hardware error condition to go potentially undetected. Therefore, extreme care should be taken when masking errors, as invalid results may be produced. It is recommended that errors only be masked during debug operation. This register may be reset by resetting the CRCU.

The interrupt mask register's bit fields are shown in Figure 10-47, and defined in Table 10-45.

Offset 0x3_F038 Access: Read/Write

| | 0–48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58–60 | 61 | 62–63 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | ICE | PE | IE | ERE | CE | KSE | DSE | ME | AE | — | IFO | — |

Figure 10-47. CRCU Interrupt Mask Register

### Table 10-45. CRCU Interrupt Mask Register Bit Definitions

| Bits | Name | Description |
|---|---|---|
| 0–48 | — | Reserved |
| 49 | ICE | Integrity Check Error. The supplied ICV(CRC) did not match the one computed by the CRCU.<br>0 Integrity check error enabled. WARNING: Do not enable this if using EU status writeback (see bits IWSE and AWSE in Section 10.4.4.1, "Channel Configuration Register (CCR)").<br>1 Integrity check error disabled |
| 50 | PE | Polynomial Error.<br>0 Polynomial error enabled<br>1 Polynomial error disabled |
| 51 | IE | Internal Error. An internal processing error was detected while performing hashing.<br>0 Internal error enabled<br>1 Internal error disabled |
| 52 | ERE | Early Read Error. The CRCU register was read while the CRCU was performing hashing.<br>0 Early read error enabled<br>1 Early read error disabled |
| 53 | CE | Context Error. The CRCU key register, the key size register, the data size register, or the mode register, was modified while the CRCU was performing hashing.<br>0 Context error enabled<br>1 Context error disabled |
| 54 | KSE | Key Size Error. A value outside the bounds was written to the CRCU key size register<br>0 Key size error enabled<br>1 Key size error disabled |
| 55 | DSE | Data Size Error. An inconsistent value was written to the CRCU data size register:<br>0 Data size error enabled<br>1 Data size error disabled |
| 56 | ME | Mode Error. An illegal value was detected in the mode register.<br>0 Mode error enabled<br>1 Mode error disabled |
| 57 | AE | Address Error. An illegal read or write address was detected within the CRCU address space.<br>0 Address error enabled<br>1 Address error disabled |
| 58–60 | — | Reserved |
| 61 | IFO | Input FIFO Overflow. The CRCU input FIFO was pushed while full.<br>0 Input FIFO overflow error enabled<br>1 Input FIFO overflow error disabled |
| 62–63 | — | Reserved |

### 10.7.3.10 CRCU ICV Size Register

The CRCU ICV size register (shown in Figure 10-48) is word readable/writable for compatibility with the MDEU. Values written to this location are always ignored, and reads from this location always returns zero. A context error is generated if this register is written after processing has begun.

|  | 0 … 63 |
|---|---|
| Field | — |
| Reset | 0 |
| R/W | R/W |
| Addr | CRCU 0x3_F040 |

Figure 10-48. CRCU ICV Size Register

### 10.7.3.11 CRCU End of Message Register

The CRCU end of message register (shown in Figure 10-49) is used to indicate that all data has been written to the CRCU (in channel-driven access, this signaling is done automatically). A write to this register is required to complete a CRC32 operation. The CRCU starts processing message data as soon as the data size register is written and data becomes available in the FIFO, but it does not process a remaining partial word or perform an ICV check until this register is written.

Any value written to this register has the same effect. Reading this register returns a zero value.

Offset 0x3_F050 Access: Write only

|  | 0 … 63 |
|---|---|
| R | ▨ |
| W |  |
| Reset | All zeros |

Figure 10-49. CRCU End of Message Register

### 10.7.3.12 CRCU Context Register

This register can be written with an intermediate CRC result or desired initial state prior to processing any data. Once processing is complete, the CRC result is available from this register. Note that the CRC result is stored in the upper half of this register; the lower half is not used. The reset state of this register is all ones, as this allows the CRC32 algorithm to detect bit errors in the leading zeros of a message. Figure 10-50 shows the bit position of each term in the written context value. A context error is generated if this register is written after processing has begun. An early read error is generated if this register is read while the module is busy.

Offset 0x3_F100 Access: Read/Write

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | $x^{31}$ | $x^{30}$ | $x^{29}$ | $x^{28}$ | $x^{27}$ | $x^{26}$ | $x^{25}$ | $x^{24}$ | $x^{23}$ | $x^{22}$ | $x^{21}$ | $x^{20}$ | $x^{19}$ | $x^{18}$ | $x^{17}$ | $x^{16}$ | $x^{15}$ | $x^{14}$ | $x^{13}$ | $x^{12}$ | $x^{11}$ | $x^{10}$ | $x^{9}$ | $x^{8}$ | $x^{7}$ | $x^{6}$ | $x^{5}$ | $x^{4}$ | $x^{3}$ | $x^{2}$ | $x^{1}$ | $x^{0}$ |
| Reset | All zeros | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

| | 32–63 |
|---|---|
| R<br>W | — |
| Reset | All zeros |

**Figure 10-50. CRCU Context Register (Write)**

If the CRCU is in the default output mode (DOS and DOC bits are 0), this register holds the CRC remainder after it has been bit swapped, byte swapped, and complemented. This sequence of operations is described in the protocol specifications and generates a result, which can be written directly to the end of a frame or command. The result is shown in Figure 10-51.

Offset 0x3_F100 Access: Read/Write

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | $x^{24}$ | $x^{25}$ | $x^{26}$ | $x^{27}$ | $x^{28}$ | $x^{29}$ | $x^{30}$ | $x^{31}$ | $x^{16}$ | $x^{17}$ | $x^{18}$ | $x^{19}$ | $x^{20}$ | $x^{21}$ | $x^{22}$ | $x^{23}$ | $x^{8}$ | $x^{9}$ | $x^{10}$ | $x^{11}$ | $x^{12}$ | $x^{13}$ | $x^{14}$ | $x^{15}$ | $x^{0}$ | $x^{1}$ | $x^{2}$ | $x^{3}$ | $x^{4}$ | $x^{5}$ | $x^{6}$ | $x^{7}$ |
| Reset | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

**Figure 10-51. CRCU Context Register, Upper Half (Read in Default Mode)**

If the CRCU is in the disable output swap mode (DOS bit is 1), this register holds the unswapped but complemented CRC remainder, as shown in Figure 10-52.

Offset 0x3_F100 Access: Read/Write

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | $x^{31}$ | $x^{30}$ | $x^{29}$ | $x^{28}$ | $x^{27}$ | $x^{26}$ | $x^{25}$ | $x^{24}$ | $x^{23}$ | $x^{22}$ | $x^{21}$ | $x^{20}$ | $x^{19}$ | $x^{18}$ | $x^{17}$ | $x^{16}$ | $x^{15}$ | $x^{14}$ | $x^{13}$ | $x^{12}$ | $x^{11}$ | $x^{10}$ | $x^{9}$ | $x^{8}$ | $x^{7}$ | $x^{6}$ | $x^{5}$ | $x^{4}$ | $x^{3}$ | $x^{2}$ | $x^{1}$ | $x^{0}$ |
| Reset | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

**Figure 10-52. CRCU Context Register, Upper Half (Read in DOS Mode)**

If the CRCU is in the disable output complement mode (DOC bit is 1), this register holds the uncomplemented but swapped CRC remainder, as shown in Figure 10-53.

Offset 0x3_F100 Access: Read/Write

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | $x^{24}$ | $x^{25}$ | $x^{26}$ | $x^{27}$ | $x^{28}$ | $x^{29}$ | $x^{30}$ | $x^{31}$ | $x^{16}$ | $x^{17}$ | $x^{18}$ | $x^{19}$ | $x^{20}$ | $x^{21}$ | $x^{22}$ | $x^{23}$ | $x^{8}$ | $x^{9}$ | $x^{10}$ | $x^{11}$ | $x^{12}$ | $x^{13}$ | $x^{14}$ | $x^{15}$ | $x^{0}$ | $x^{1}$ | $x^{2}$ | $x^{3}$ | $x^{4}$ | $x^{5}$ | $x^{6}$ | $x^{7}$ |
| Reset | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

**Figure 10-53. CRCU Context Register, Upper Half (Read in DOC Mode)**

If the CRCU is in both disable output complement mode (DOC bit is 1) and disable output swap mode (DOS bit is 1), this register holds the uncomplemented and unswapped CRC remainder, as shown in Figure 10-54. This form is the one used internally to match against the polynomial specific residue when performing ICV checking. This form can also be written back to the CRCU after reset to continue a partial CRC operation.

Offset 0x3_F100 Access: Read/Write

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | $x^{31}$ | $x^{30}$ | $x^{29}$ | $x^{28}$ | $x^{27}$ | $x^{26}$ | $x^{25}$ | $x^{24}$ | $x^{23}$ | $x^{22}$ | $x^{21}$ | $x^{20}$ | $x^{19}$ | $x^{18}$ | $x^{17}$ | $x^{16}$ | $x^{15}$ | $x^{14}$ | $x^{13}$ | $x^{12}$ | $x^{11}$ | $x^{10}$ | $x^{9}$ | $x^{8}$ | $x^{7}$ | $x^{6}$ | $x^{5}$ | $x^{4}$ | $x^{3}$ | $x^{2}$ | $x^{1}$ | $x^{0}$ |
| Reset | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

**Figure 10-54. CRCU Context Register, Upper Half (Read in DOC and DOS Modes)**

## 10.7.3.13 CRCU Key Register

The CRCU key register stores the polynomial and residue for the dynamic custom mode as set in the mode register (see Section 10.7.3.2, "CRCU Mode Register"). Figure 10-55 shows the bit position of each coefficients. The reset value of this register is all zeros with a one in bit position 0. This register is dynamic, in that it is reset by performing a re-initialize or a software reset. This allows a custom polynomial to be used for specific processing without changing the platform-specific static custom polynomial stored in the control register (see Section 10.7.3.6, "CRCU Control Register"). A residue does not need to be programmed unless ICV checking is being performed. A context error is generated if this register is written after processing has begun. A polynomial error is generated if a value is written to this register which does not have a one in bit position 0.

Offset 0x3_F400 Access: Read/Write

| | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | $r^{31}$ | $r^{30}$ | $r^{29}$ | $r^{28}$ | $r^{27}$ | $r^{26}$ | $r^{25}$ | $r^{24}$ | — | | $r^{7}$ | $r^{6}$ | $r^{5}$ | $r^{4}$ | $r^{3}$ | $r^{2}$ | $r^{1}$ | $r^{0}$ |
| | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 |
| R<br>W | $g^{31}$ | $g^{30}$ | $g^{29}$ | $g^{28}$ | $g^{27}$ | $g^{26}$ | $g^{25}$ | $g^{24}$ | — | | $g^{7}$ | $g^{6}$ | $g^{5}$ | $g^{4}$ | $g^{3}$ | $g^{2}$ | $g^{1}$ | $g^{0}$ |

**Figure 10-55. CRCU Key Register**

## 10.7.3.14 CRCU FIFO

Words written to this address range are pushed onto the CRCU input FIFO, thereby buffering them for processing. Partial words and misaligned data can be written to this address and it is automatically realigned based on a big endian byte order.

## 10.7.4 Data Encryption Standard Execution Unit (DEU)

This section contains details about the Data Encryption Standard execution unit (DEU), including modes of operation, status and control registers, and FIFOs.

Most of the registers described here would not normally be accessed by the host. They are documented here mainly for debug purposes. In typical operation, the DEU is used through channel-controlled access, which means that most reads and writes of DEU registers are directed by the SEC channels. Driver software would perform host-controlled register accesses only on a few registers for initial configuration and error handling.

### 10.7.4.1 DEU Mode Register

The DEU mode register contains 3 bits which are used to program DEU operation.

The mode register is cleared when the DEU is reset or re-initialized. Setting a reserved mode bit generates a data error. If the mode register is modified during processing, a context error is generated.

Offset 0x3_2000 Access: Read/Write

| | 0 … 59 | 60 61 | 62 | 63 |
|---|---|---|---|---|
| R<br>W | — | CM | TS | ED |
| Reset | All zeros | | | |

**Figure 10-56. DEU Mode Register**

Table 10-46 describes DEU mode register fields.

**Table 10-46. DEU Mode Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| The following bits are described for information only. They are not under direct user control. | | |
| 0–55 | — | Reserved |
| The following bits are controlled through the MODE0 field of the descriptor header. | | |
| 56–59 | — | Reserved |
| 60–61 | CM | Cipher Mode: Used to define the mode of DEU operation. See Table 10-47 for mode bit combinations. |
| 62 | TS | Triple/Single DES. If set, DEU operates the Triple DES algorithm; if not set, DEU operates the single DES algorithm.<br>0 Single DES<br>1 Triple DES |
| 63 | ED | Encrypt/decrypt. If set, DEU operates the encryption algorithm; if not set, DEU operates the decryption algorithm.<br>0 Perform decryption<br>1 Perform encryption |

**Table 10-47. DEU Cipher Modes**

| Mode | CM (60:61) |
|---|---|
| ECB | 00 |
| CBC | 01 |

**Table 10-47. DEU Cipher Modes (continued)**

| Mode | CM (60:61) |
|---|---|
| CFB-64 | 10 |
| OFB-64 | 11 |

### 10.7.4.2 DEU Key Size Register

This value indicates the number of bytes of key memory that should be used in encrypting or decrypting. If the DEU mode register is set for single DES, any value other than 8 bytes automatically generates a key size error in the DEU interrupt status register. If the mode bit is set for triple DES, any value other than 16 bytes (112 bits for 2-key triple DES (K1=K3) or 24 bytes (168 bits for 3-key triple DES) generates an error. Triple DES always uses K1 to encrypt, K2 to decrypt, K3 to encrypt.

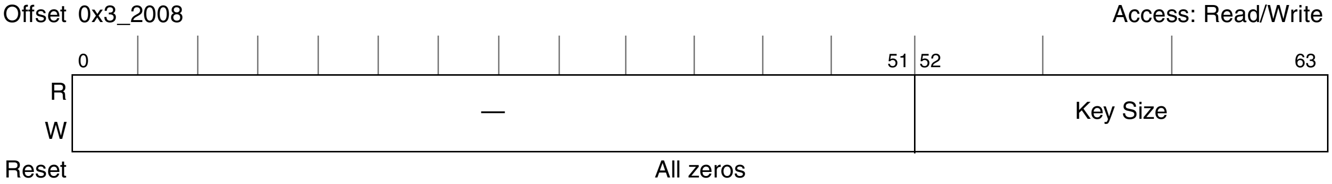


**Figure 10-57. DEU Key Size Register**

Table 10-48 shows the legal values for DEU key size.

**Table 10-48. DEU Key Size Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–51 | — | Reserved |
| 52–63 | Key Size | 8 bytes = 0x08 (only legal value if mode is single DES.)<br>16 bytes = 0x10 (for 2 key 3DES, K1 = K3)<br>24 bytes = 0x18 (for 3 key 3DES) |

### 10.7.4.3 DEU Data Size Register (DEUDSR)

The DEUDSR, shown in Figure 10-58, is used to verify that the data to be processed by the DEU is divisible by the DES algorithm block size of 64 bits. The DEU does not automatically pad messages out to 64-bit blocks; therefore, any message processed by the DEU must be divisible by 64 bits or a data size error will occur.

In channel-driven operation, the full message length (data size) to be encrypted or decrypted by the DEU is copied from the descriptor to the DEUDSR; however, only bits 58–63 are checked to determine if there is a data size error. If bits 58–63 are all zeros, the message is evenly divisible into 64-bit blocks. In host-driven operation,the user must write the data size to the DEUDSR. If bits 58–63 are not all zero, then a data size error occurs.This register is cleared when the DEU is reset or re-initialized.

Offset 0x3_2008 Access: Read/Write

| | 0 … 51 | 52 … 63 |
|---|---|---|
| R<br>W | — | Data Size |
| Reset | All zeros | |

Figure 10-58. DEU Data Size Register

### 10.7.4.4 DEU Reset Control Register

This register, shown in Figure 10-59, allows three levels reset of just DEU, as defined by the three self-clearing bits:

| | 0 … 60 | 61 | 62 | 63 |
|---|---|---|---|---|
| Field | — | RI | MI | SR |
| Reset | 0 | | | |
| R/W | R/W | | | |
| Addr | DEU 0x3_2018 | | | |

Figure 10-59. DEU Reset Control Register

Table 20-72 describes DEU reset control register fields.

Table 10-49. DEU Reset Control Register Field Descriptions

| Bits | Names | Description |
|---|---|---|
| 0–60 | — | Reserved |
| 61 | RI | Reset interrupt. Writing this bit active high causes DEU interrupts signaling done and error to be reset. It further resets the state of the DEU interrupt status register.<br>0 Do not reset<br>1 Reset interrupt logic |
| 62 | MI | Module initialization is nearly the same as software reset, except that the interrupt mask register remains unchanged. this module initialization includes execution of an initialization routine, completion of which is indicated by the RESET_DONE bit in the DEU status register<br>0 Do not reset<br>1 Reset most of DEU |
| 63 | SR | Software reset is functionally equivalent to hardware reset (the RESET# pin), but only for DEU. All registers and internal state are returned to their defined reset state. Upon negation of SW_RESET, the DEU enters a routine to perform proper initialization of the parameter memories. The RESET_DONE bit in the DEU status register indicates when this initialization routine is complete<br>0 Do not reset<br>1 Full DEU reset |

### 10.7.4.5 DEU Status Register

This status register, displayed in Figure 10-60, contains 6 fields which reflect the state of DEU internal signals. The DEU status register is read only. Writing to this location results in address error being reflected in the DEU interrupt status register.

Offset 0x3_2028 Access: Read only

| | 0 39 | 40 47 | 48 55 | 56 57 | 58 | 59 60 | 61 | 62 | 63 |
|---|---|---|---|---|---|---|---|---|---|
| R | — | OFL | IFL | — | HALT | — | EI | DI | RD |
| W | | | | | | | | | |
| Reset | All zeros | | | | | | | | |

**Figure 10-60. DEU Status Register**

Table 10-50 describes the DEU status register's bit settings.

**Table 10-50. DEU Status Register**

| Bits | Name | Description |
|---|---|---|
| 0–39 | — | Reserved |
| 40-47 | OFL | The number of dwords currently in the output FIFO |
| 48-55 | IFL | The number of dwords currently in the input FIFO |
| 56-57 | — | Reserved |
| 58 | HALT | Halt. Indicates that the DEU has halted due to an error.<br>0 DEU not halted<br>1 DEU halted<br>**Note:** Because the error causing the DEU to stop operating may be masked before reaching the interrupt status register, the DEU interrupt status register is used to provide a second source of information regarding errors preventing normal operation. |
| 59-60 | — | Reserved |
| 61 | EI | Error interrupt: This status bit reflects the state of the error interrupt signal, as sampled by the controller interrupt status register (Section 10.5.4.2.2, "Interrupt Status Register (ISR)").<br>0 DEU is not signaling error<br>1 DEU is signaling error |
| 62 | DI | Done interrupt: This status bit reflects the state of the done interrupt signal, as sampled by the controller interrupt status register (Section 10.5.4.2.2, "Interrupt Status Register (ISR)").<br>0 DEU is not signaling done<br>1 DEU is signaling done |
| 63 | RD | Reset done. This status bit, when high, indicates that DEU has completed its reset sequence, as reflected in the signal sampled by the appropriate channel.<br>0 Reset in progress<br>1 Reset done<br>**Note:** Reset Done resets to 0, but has typically switched to 1 by the time a user checks the register, indicating the EU is ready for operation. |

### 10.7.4.6 DEU Interrupt Status Register

The DEU interrupt status register indicates which unmasked errors have occurred and have generated error interrupts to the channel. Each bit in this register can only be set if the corresponding bit of the DEU interrupt mask register is zero (see Section 10.7.4.7, “DEU Interrupt Mask Register”).

If the DEU interrupt status register is non-zero, the DEU halts and the DEU error interrupt signal is asserted to the controller (see Section 10.5.4.2.2, “Interrupt Status Register (ISR)”). In addition, if the DEU is being operated through channel-controlled access, then an interrupt signal is generated to the channel to which this EU is assigned. The EU error then appears in bit 55 of the channel status register (see Table 10-15) and generates a channel error interrupt to the controller.

If the interrupt status register is written from the host, 1s in the value written are recorded in the interrupt status register if the corresponding bit is unmasked in the interrupt mask register. All other bits are cleared. This register can also be cleared by setting the RI bit of the DEU reset control register.

The definition of each bit in the DEU interrupt status register is shown in Figure 10-61.

Offset DEU 0x3_2030 Access: Read only

| | 0 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | — | | KPE | IE | ERE | CE | KSE | DSE | ME | AE | OFE | IFE | IFU | IFO | OFU | OFO |
| W | | | | | | | | | | | | | | | | |
| Reset | All zeros | | | | | | | | | | | | | | | |

**Figure 10-61. DEU Interrupt Status Register**

Table 10-51 describes DEU interrupt status register fields.

**Table 10-51. DEU Interrupt Status Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0-49 | — | Reserved |
| 50 | KPE | Key Parity Error. Defined parity bits in the keys written to the key registers did not reflect odd parity correctly. (Note that key register 2 and key register 3 are checked for parity only if the appropriate DEU mode register bit indicates triple DES. Also, key register 3 is checked only if key size reg = 24. Key register 2 is checked only if key size reg = 16 or 24.)<br>0 No error detected<br>1 Key parity error |
| 51 | IE | Internal Error. An internal processing error was detected while performing encryption.<br>0 No error detected<br>1 Internal error<br>**Note:** This bit is asserted any time an enabled error condition occurs and can only be cleared by setting the corresponding bit in the interrupt mask register or by resetting the DEU. |
| 52 | ERE | Early Read Error. The DEU IV register was read while the DEU was performing encryption.<br>0 No error detected<br>1 Early read error |
| 53 | CE | Context Error. A DEU key register or the key size register, data size register, mode register, or IV register was modified while DEU was performing encryption.<br>0 No error detected<br>1 Context error |

### Table 10-51. DEU Interrupt Status Register Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 54 | KSE | Key Size Error. An inappropriate value (8 being appropriate for single DES, and 16 and 24 being appropriate for triple DES) was written to the DEU key size register<br>0 No error detected<br>1 Key size error |
| 55 | DSE | Data Size Error (DSE): A value was written to the DEU data size register that is not a multiple of 64 bits if ECB, CBC, or CFB mode is selected. If OFB mode is selected, any data size value is permitted.<br>0 No error detected<br>1 Data size error |
| 56 | ME | Mode error. An illegal value was detected in the mode register. Note: writing to reserved bits in mode register is likely source of error.<br>0 No error detected<br>1 Mode error |
| 57 | AE | Address error. An illegal read or write address was detected within the DEU address space.<br>0 No error detected<br>1 Address error |
| 58 | OFE | Output FIFO error. The DEU output FIFO was detected non-empty upon write of DEU data size register.<br>0 No error detected<br>1 Output FIFO non-empty error |
| 59 | IFE | Input FIFO error. The DEU input FIFO was detected non-empty upon generation of done interrupt.<br>0 No error detected<br>1 Input FIFO non-empty error |
| 60 | IFU | Input FIFO Underflow. The DEU input FIFO was read while empty.<br>0 No error detected<br>1 Input FIFO has had underflow error |
| 61 | IFO | Input FIFO Overflow. The DEU input FIFO was pushed while full.<br>0 No error detected<br>1 Input FIFO has overflowed<br>**Note:** When operated through channel-controlled access, the SEC implements flow control, and FIFO size is not a limit to data input. When operated through host-controlled access, the DEU cannot accept FIFO inputs larger than 256 bytes without overflowing. |
| 62 | OFU | Output FIFO Underflow. The DEU output FIFO was read while empty.<br>0 No error detected<br>1 Output FIFO has underflow error |
| 63 | OFO | Output FIFO Overflow. The DEU output FIFO was pushed while full.<br>0 No error detected<br>1 Output FIFO has overflowed |

### 10.7.4.7 DEU Interrupt Mask Register

The DEU interrupt mask register, controls the setting of bits in the DEU interrupt status register, as described in Section 10.7.4.6, “DEU Interrupt Status Register”. If a DEU interrupt mask register bit is set, then the corresponding interrupt status register bit is always zero.

Masking an error bit allows for a hardware error condition to go potentially undetected. Therefore, extreme care should be taken when masking errors, as invalid results may be produced. It is recommended that errors only be masked during debug operation. This register may be reset by resetting the DEU.

Offset 0x3_2038 Access: Read/Write

| | 0 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | | KPE | IE | ERE | CE | KSE | DSE | ME | AE | OFE | IFE | IFU | IFO | OFU | OFO | KPE |
| Reset | 1000 | | | | | | | | | | | | | | | | |

**Figure 10-62. DEU Interrupt Mask Register**

**Table 10-52. DEU Interrupt Mask Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0-49 | — | Reserved |
| 50 | KPE | Key Parity Error. The defined parity bits in the keys written to the key registers did not reflect odd parity correctly. (Note that key register 2 and key register 3 are only checked for parity if the appropriate DEU mode register bit indicates triple DES.<br>0 Key parity error enabled<br>1 Key parity error disabled |
| 51 | IE | Internal Error. An internal processing error was detected while performing encryption.<br>0 Internal error enabled<br>1 Internal error disabled |
| 52 | ERE | Early Read Error. The DEU IV Register was read while the DEU was performing encryption.<br>0 Early read error enabled<br>1 Early read error disabled |
| 53 | CE | Context Error. A DEU key register,or the key size register, the data size register, the mode register, or IV register was modified while DEU was performing encryption.<br>0 Context error enabled<br>1 Context error disabled |
| 54 | KSE | Key Size Error. An inappropriate value (8 being appropriate for single DES, and 16 and 24 being appropriate for Triple DES) was written to the DEU key size register<br>0 Key size error enabled<br>1 Key size error disabled |
| 55 | DSE | Data Size Error (DSE). A value that is not a multiple of 64 bits was written to the DEU data size register if ECB, CBC, or CFB mode is selected. If OFB mode is selected, any data size value is permitted.<br>0 Data size error enabled<br>1 Data size error disabled |
| 56 | ME | Mode Error. An illegal value was detected in the mode register.<br>0 Mode error enabled<br>1 Mode error disabled |

Table 10-52. DEU Interrupt Mask Register Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 57 | AE | Address Error. An illegal read or write address was detected within the DEU address space.<br>0 Address error enabled<br>1 Address error disabled |
| 58 | OFE | Output FIFO Error. The DEU output FIFO was detected non-empty upon write of DEU data size register<br>0 Output FIFO non-empty error enabled<br>1 Output FIFO non-empty error disabled |
| 59 | IFE | Input FIFO Error. The DEU input FIFO was detected non-empty upon generation of done interrupt<br>0 Input FIFO non-empty error enabled<br>1 Input FIFO non-empty error disabled |
| 60 | IFU | Input FIFO Underflow. The DEU input FIFO was read while empty.<br>0 Input FIFO Underflow error enabled<br>1 Input FIFO Underflow error disabled |
| 61 | IFO | Input FIFO Overflow. The DEU input FIFO was pushed while full.<br>0 Input FIFO overflow error enabled<br>1 Input FIFO overflow error disabled<br>**Note:** When operated through channel-controlled access, the SEC implements flow control, and FIFO size is not a limit to data input. When operated through host-controlled access, the DEU cannot accept FIFO inputs larger than 256 bytes without overflowing. |
| 62 | OFU | Output FIFO Underflow. The DEU output FIFO was read while empty.<br>0 Output FIFO underflow error enabled<br>1 Output FIFO underflow error disabled |
| 63 | OFO | Output FIFO Overflow. The DEU output FIFO was pushed while full.<br>0 Output FIFO Overflow error enabled<br>1 Output FIFO Overflow error disabled |

### 10.7.4.8 DEU End of Message Register

The DEU end of message register, shown in Figure 10-63, is used to signal to the DEU that the final message block has been written to the input FIFO (in channel-driven access, this signaling is done automatically). The DEU will not process the last block of data in its input FIFO until this register is written. Once the end of message register is written, the DEU processes any remaining data in the input FIFO and generates the done interrupt.

The value written to this register does not matter. A read of this register always returns a zero value.

Offset 0x3_2050 Access: Write only

| | 0–63 |
|---|---|
| R<br>W | — |
| Reset | All zeros |

Figure 10-63. DEU End of Message Register

### 10.7.4.9 DEU IV Register

For CBC mode, the initialization vector is written to and read from the DEU IV register. The value of this register changes as a result of the encryption process and reflects the context of DEU. Reading this memory location while the module is processing data generates an error interrupt.

### 10.7.4.10 DEU Key Registers

The DEU uses three write-only key registers, K1, K2, and K3, to perform encryption and decryption. In Single DES mode, only K1 may be written. The value written to K1 is simultaneously written to K3, auto-enabling the DEU for 112-bit Triple DES if the key size register indicates 2 key 3DES is to be performed (key size = 16 bytes). To operate in 168-bit Triple DES, K1 must be written first, followed by the write of K2, then K3.

Reading any of these memory locations generates an address error interrupt.

### 10.7.4.11 DEU FIFOs

DEU uses an input FIFO/output FIFO pair to hold data before and after the encryption process. Normally, the channels control all access to these FIFOs. For host-controlled operation, a write to anywhere in the DEU FIFO address space enqueues data to the DEU input FIFO, and a read from anywhere in the DEU FIFO address space dequeues data from the DEU output FIFO.

Writes to the input FIFO go first to a staging register which can be written by byte, word (4 bytes), or dword (8 bytes). When all 8 bytes of the staging register have been written, the entire dword is automatically enqueued into the FIFO. If any byte is written twice between enqueues, it causes an error interrupt of type AE from the EU. Since the DEU data length should always be a multiple of 8 bytes, the last write should complete a dword. However, if there is any partial dword in the staging register when the DEU end of message register is written, the partial dword is automatically padded with zeros to a full 8 bytes and enqueued to the input FIFO.

The output FIFO is readable by byte, word, or dword. When all 8 bytes of the head dword have been read, that dword is automatically dequeued from the FIFO so that the next dword (if any) becomes available for reading. If any byte is read twice between dequeues, it causes an error interrupt of type AE from the EU.

Overflows and underflows caused by reading or writing the DEU FIFOs are reflected in the DEU interrupt status register.

## 10.7.5 Kasumi Execution Unit (KEU)

This section contains details about the Kasumi execution unit (KEU), including modes of operation, status and control registers, and FIFOs. The KEU has been designed to support the f8 confidentiality function of the 3GPP, GSM A5/3, EDGE A5/3, and GPRS GEA3 algorithms. The KEU also supports the 3GPP f9 integrity function.

Most of the registers described here would not normally be accessed by the host. They are documented here mainly for debug purpose. In typical operation, the KEU is used through channel-controlled access, which means that most reads and writes of the KEU registers are directed by the SEC channels. Driver software performs host-controlled register accesses only on a few registers for initial configuration and error handling.

This execution unit (EU) includes an ICVchecking feature, which means it can generate an ICV and compare it to another supplied ICV. The pass/fail result of this ICV check can be returned to the host either through interrupt or by using a writeback of EU status fields into the host memory, but not using both methods at the same time.

To signal the ICV checking result by status writeback, turn on either the IWSE bit or AWSE bit in the channel configuration register (for more information, see Section 10.4.4.1, “Channel Configuration Register (CCR)”), and mask the ICE bit in the interrupt mask register (Section 10.7.5.7, “KEU Interrupt Mask Register (KEUIMR)”). In this case the normal DONE signal (by interrupt or writeback) is undisturbed.

To signal the ICV checking result by interrupt, unmask the ICE bit in the interrupt mask register and turn off the IWSE and AWSE bits in the channel configuration register. If there is no ICV mismatch, the normal DONE signal (by interrupt or writeback) occurs. When there is an ICV mismatch, there is an ERROR interrupt signal to the host, but no DONE interrupt signal or writeback.

### 10.7.5.1 KEU Mode Register (KEUMR)

The KEU mode register, shown in Figure 10-64, contains several bits which are used to program the KEU. The mode register is cleared when the KEU is reset or re-initialized. Setting a reserved mode bit generates a data error. Setting both the GSM and EDGE bits to one generates a data error. If the KEU mode register is modified during processing, a context error is generated.

Offset 0x3_E000 Access: Read/Write

| | 0–55 | 56 | 57 | 58 | 59 | 60 | 61 | 62–63 |
|---|---|---|---|---|---|---|---|---|
| R<br>W | — | GSM | CICV | EDGE | PE | INT | — | ALG |
| Reset | All zeros | | | | | | | |

**Figure 10-64. KEU Mode Register**

Table 10-53 describes the KEU mode register fields.

**Table 10-53. KEU Mode Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–55 | — | Reserved |
| 56 | GSM | Select GSM A5/3 blocks<br>0 GSM A5/3 blocks not selected<br>1 GSM A5/3 blocks selected<br>**Note 1:** For GSM A5/3, Two 114-bit blocks are required to be produced each 4.615mS slot. If GSM = 1, the first read of the output FIFO retrieves the first 64 bits of block 1. The second read of the output FIFO retrieves the next 50 bits of block 1 (the remaining bits of this 64-bit word are cleared to zero). The third read of the output FIFO retrieves the first 64 bits of block 2, while a fourth read of the output FIFO retrieves the next 50 bits of block 2 (the remaining bits of this 64-bit word are cleared to zero).<br>**Note 2:** If GSM = 0, 228 contiguous bits may be read with successive reads of the output FIFO. In this case the host (application) is responsible for handling the A5/3 block formatting.<br>**Note 3:** If GSM is set to 1, while EDGE = 1, an interrupt/error is generated. |

Table 10-53. KEU Mode Register Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 57 | CICV | Compare integrity check values.<br>0 Normal operation; no ICV comparison.<br>1 After the ICV is computed, compare it to the data in the KEU's ICV_In register. If the ICVs do not match, send an error interrupt to the channel. Only applicable when the ALG field is set to a function that uses f9. |
| 58 | EDGE | Select EDGE A5/3 blocks<br>0 EDGE A5/3 blocks not selected<br>1 EDGE A5/3 blocks selected<br>**Note 1:** For EDGE A5/3, two 348-bit blocks are required to be produced each 4.615mS slot. If EDGE = 1, the first five reads of the output FIFO retrieve the first 320 bits of block 1. The sixth read of the output FIFO retrieves the final 28 bits of block 1 (the remaining bits of the sixth 64-bit word are cleared to zero). The next five reads of the output FIFO retrieve the first 320 bits of block 2. The following read of the output FIFO retrieves the final 28 bits of block 2 (the remaining bits of this 64-bit word are cleared to zero).<br>**Note 2:** If EDGE = 0, 696 contiguous bits may be read with successive reads of the output FIFO. In this case the host (application) is responsible for handling the A5/3 block formatting.<br>**Note 3:** If EDGE is set to 1, whilst GSM = 1, an interrupt/error is generated. |
| 59 | PE | Process end of message. Enables final processing of last message block (f9 only).<br>0 Prevent final block processing (message incomplete)<br>1 Enable final block processing (message complete)<br>**Note:** PE is closely connected with the KEU data size register, see Section 10.7.5.3, "KEU Data Size Register (KEUDSR)" for more details. |
| 60 | INT | Initialization. Enables initialization for a new message.<br>0 Prevent initialization<br>1 Enable initialization<br>**Note:** For f8 or f9 operations, if the 3G frame (or message) is being processed through a single descriptor, the Initialization bit should be set. If the frame is split across multiple descriptors, this bit should only be set in the descriptor that processes the first block of the message. |
| 61 | — | Reserved |
| 62–63 | ALG | Algorithm selection. Specifies the functions to perform.<br>00 Perform f8 function only<br>01 Reserved<br>10 Perform f9 function only<br>11 Reserved |

## 10.7.5.2 KEU Key Size Register (KEUKSR)

The KEU key size register, shown in Figure 10-65, stores the number of bytes in the key. It should be set to 16 bytes. This register is cleared when the KEU is reset or re-initialized. If a key size is specified that does not match the selected algorithm(s), an illegal key size error is generated.

|  | 0 … 51 | 52 … 63 |
|---|---|---|
| Field | — | Key Size (Bytes) |
| Reset | 0 |  |
| R/W | R/W |  |
| Addr | KEU 0x3_E008 |  |

Figure 10-65. KEU Key Size Register

### 10.7.5.3 KEU Data Size Register (KEUDSR)

The KEU data size register (shown in Figure 10-66) stores the number of bits to process in the final message word. As Kasumi allows for bit level granularity for encryption/decryption, there are no illegal data sizes. The proper bit length of the message must be written to notify the KEU of any padding performed by the host. This register is cleared when the KEU is reset or re-initialized.

Writing to this register signals the KEU to start processing data from the input FIFO as soon as it is available. If the value of data size is modified during processing, a context error is generated.

Kasumi processing is determined by both the data size and the setting of the process end of message (PE) bit in the KEU mode register. The PE bit determines how the final block of message data is processed. In typical descriptor-based operations, the data size register is loaded with values which are an integral number of bytes. For descriptor based f8 operations, the software is responsible for padding the data to the next byte boundary, and for removing this padding from the KEU's output. The output of the KEU is an integral number of bytes, as specified in the descriptor, automatically truncating any internal padding required to process the final 64 bits message block. As the KEU can infer when it has reached the final 64 bits message block from the length fields in the descriptor, setting the PE bit through the descriptor header is not required. While performing f8 operations, the KEU's output is the same irrespective of the setting of the PE bit.

For the descriptor-based f9 operations, the PE bit must be set through the descriptor header whenever the descriptor is being used to process the final message block. This causes the KEU to automatically pad the final block before calculating the f9 MAC.

Offset 0xE010 Access: Read/Write

| | 0 … 51 | 52 … 63 |
|---|---|---|
| R<br>W | — | Data Size (bits) |
| Reset | All zeros | |

**Figure 10-66. KEU Data Size Register**

The details of data size register and the PE interaction are more relevant when operating the KEU in direct access (slave) mode, rather than using descriptors. Note that operating the KEU in direct access mode is not recommended other than for debug test cases, and the information provided here regarding the use of data size and PE in direct access mode is to explain the behaviors that might be encountered in direct access debug operations.

PE has the following effects for direct access mode f8 operations, using the example of a 64-bit f8 keystream '0x1234567890abcdef' and the data size register containing '0x0a' (10 bits = 1 byte + 2 bits):

- PE = 0: The final ten message bits are XORed with the entire last 64-bit block of keystream, which produces 54 additional non-zero bits after the end of the real message. These additional 54 bits must be removed by the software.
- PE = 1: The final ten message bits are XORed with ten bits of keystream '0x120', and no additional bits of the false message are produced.

For direct access mode f9 operations, assertion of PE enables f9 algorithm-specified padding per the value in the data size register. If this direct access mode operation includes the final message block, then PE must be set in the KEU mode register as follows:

- PE = 0: The KEU does not perform padding. For the KEU to perform the correct computation, the last block of the message provided must be a complete 64-bit block, and the value written into the KEU data size register must indicate a multiple of 64 bits.
- PE = 1: The KEU pads the final 64 bits of the message as specified in the 3GPP f9 algorithm. The communication direction (CD) bit (see Table 10-58 for KEU IV_1 Register Fields Description) and '1' is appended to the end of the message. The result is zero-padded to 64 bits.

  For example, if the last block is xF81A_0000_0000_0000 (big endian) and data size contains 0x0F (15 bits = 1 byte + 7 bits), the word (0xF81A = 1111_1000_0001_1010), the most-significant 15 bits (underlined) are padded left to right as 1111_1000_0001_101$_1000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000, where $ is the value of the CD bit in the mode register.

### 10.7.5.4 KEU Reset Control Register (KEURCR)

The KEU reset control register, shown in Figure 10-67, allows three levels of reset of the KEU, as defined by the three self-clearing bits.

| | 0 | 60 | 61 | 62 | 63 |
|---|---|---|---|---|---|
| Field | — | | CLI | RI | SR |
| Reset | 0 | | | | |
| R/W | R/W | | | | |
| Addr | KEU 0xE018 | | | | |

**Figure 10-67. KEU Reset Control Register**

Table 10-54 describes the KEU reset control register fields.

**Table 10-54. KEU Reset Control Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–60 | — | Reserved |
| 61 | CLI | Clear Interrupts.<br>Setting this bit causes the KEU interrupt signals—DONE and ERROR—to be reset. It further resets the state of the KEU interrupt status register.<br>0 Normal operation<br>1 Clear interrupts and the KEU interrupt status register |

**Table 10-54. KEU Reset Control Register Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 62 | RI | Re- Initialization.<br>It is same as software reset (SR), except that the interrupt mask register remains unchanged. Completion of re-initialization is indicated by the RESET_DONE bit in the KEU status register.<br>0 Normal operation<br>1 Re-initialize the KEU |
| 63 | SR | Software reset.<br>Functionally equivalent to hardware reset (the $\overline{\text{RESET}}$ signal), but only for the KEU. All registers and internal state are returned to their defined reset state. Upon negation of the SR bit, the KEU enters a routine to perform proper initialization of the parameter memories. The reset done (RD) bit in the KEU status register indicates when this initialization is complete<br>0 Normal operation<br>1 Full KEU reset |

## 10.7.5.5 KEU Status Register (KEUSR)

The KEU status register, shown in Figure 10-68, is a read-only register that reflects the state of six status outputs. While writing to this location, an address error is reflected in the KEU interrupt status register.

| | 0 – 39 | 40 – 47 | 48 – 55 | 56 – 57 | 58 | 59 – 60 | 61 | 62 | 63 |
|---|---|---|---|---|---|---|---|---|---|
| Field | — | OFL | IFL | — | HALT | ICCR | EI | DI | RD |
| Reset | 0 | | | | | | | | |
| R/W | R | | | | | | | | |
| Addr | KEU 0x3_E028 | | | | | | | | |

**Figure 10-68. KEU Status Register**

Table 10-55 describes the KEU status register fields.

**Table 10-55. KEU Status Register Fields Description**

| Bits | Name | Description |
|---|---|---|
| 0–39 | — | Reserved |
| 40–47 | OFL | Output FIFO level. The number of dwords currently in the output FIFO. |
| 48–55 | IFL | Input FIFO level. The number of dwords currently in the input FIFO. |
| 56–57 | — | Reserved |
| 58 | HALT | Indicates when the KEU core has halted due to an error.<br>0 KEU not halted<br>1 KEU core halted (must be reset/re-initialized)<br>**Note:** As the error causing the KEU to stop operating may be masked to the interrupt status register, the status register is used to provide a second source of information regarding errors preventing normal operation. |

Table 10-55. KEU Status Register Fields Description (continued)

| Bits | Name | Description |
|---|---|---|
| 59-60 | ICCR | Integrity check comparison result.<br>00 No integrity check comparison was performed.<br>01 The integrity check comparison passed.<br>10 The integrity check comparison failed.<br>11 Reserved<br>**Note:** A passed or failed result is generated only if ICV checking is enabled and the algorithm selected is f9. |
| 61 | EI | Error interrupt. Reflects the state of the ERROR interrupt signal, as sampled by the controller interrupt status register (Section 10.5.4.2.2, "Interrupt Status Register (ISR)").<br>0 KEU is not signaling error<br>1 KEU is signaling error |
| 62 | DI | Done interrupt. Reflects the state of the DONE interrupt signal, as sampled by the controller interrupt status register (Section 10.5.4.2.2, "Interrupt Status Register (ISR)").<br>0 Processing not done<br>1 All bytes processed |
| 63 | RD | Reset done. Indicates when the KEU has completed its reset sequence, as reflected in the signal sampled by the appropriate channel.<br>0 Reset in progress<br>1 Reset done |

### 10.7.5.6 KEU Interrupt Status Register (KEUISR)

The KEU interrupt status register tracks the state of possible errors, provided those errors are not masked, through the KEU interrupt control register.

The KEU interrupt status register indicates the unmasked errors that have occurred and have generated the ERROR interrupt signals to the channel. Each bit in this register can only be set if the corresponding bit of the KEU interrupt mask register is zero (see Section 10.7.5.7, "KEU Interrupt Mask Register (KEUIMR)").

If the KEU interrupt status register is non-zero, the KEU halts and the KEU ERROR interrupt signal is asserted to the controller (see Section 10.5.4.2.2, "Interrupt Status Register (ISR)"). In addition, if the KEU is being operated through channel-controlled access, then an interrupt signal is generated to the channel to which the EU is assigned. The EU error then appears in the bit 55 of the channel pointer status register (for more information, see Table 10-15 on page 10-43) and generates a channel error interrupt to the controller.

This register can be cleared by setting the RI bit of the KEU reset control register. If a KEU error is reported by the channel while operating in descriptor mode, the user can rely on the channel to clear the KEU interrupt by writing the Continue bit in the channel configuration register (for more information, see Section 10.4.4.1, "Channel Configuration Register (CCR)"). Setting any error bit in this register causes the KEU to signal the corresponding error, unless the associated error has been masked in the KEU interrupt mask register.

The definition of each bit in the KEU interrupt status register is shown in Figure 10-69.

| | 0 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | — | ICE | — | IE | ERE | CE | KSE | DSE | DE | AE | OFE | IFE | — | IFO | OFU | — |
| Reset | 0 | | | | | | | | | | | | | | | |
| R/W | R/W | | | | | | | | | | | | | | | |
| Addr | KEU 0x3_E030 | | | | | | | | | | | | | | | |

**Figure 10-69. KEU Interrupt Status Register**

Table 10-56 describes the KEU interrupt status register signals.

**Table 10-56. KEU Interrupt Status Register Signals Description**

| Bits | Name | Description |
|---|---|---|
| 0–48 | — | Reserved |
| 49 | ICE | Integrity check error.<br>0 No error detected<br>1 Integrity check error detected. An ICV check was performed on an f9 result and the supplied ICV did not match the one computed by the KEU. |
| 50 | — | Reserved |
| 51 | IE | Internal error. An internal processing error was detected while the KEU was processing.<br>0 No error detected<br>1 Internal error<br>This bit is set any time an enabled error condition occurs and can only be cleared by setting the corresponding bit in the interrupt mask register or by resetting the KEU. |
| 52 | ERE | Early read error. A KEU context or IV register was read while the KEU was processing.<br>0 No error detected<br>1 Early read error |
| 53 | CE | Context error. A KEU key register, the key size register, the data size register, the mode register, or IV register was modified while the KEU was processing.<br>0 No error detected<br>1 Context error |
| 54 | KSE | Key size error. An inappropriate value (not 16 or 32 bytes) was written to the KEU key size register.<br>0 No error detected<br>1 Key size error |
| 55 | DSE | Data size error. A value was written to the KEU data size register that is greater than 64 bits.<br>0 No error detected<br>1 Data size error |
| 56 | DE | Data error. Invalid data was written to a register or a reserved mode bit was set.<br>0 Valid data<br>1 Reserved or invalid mode selected |
| 57 | AE | Address error. An illegal read or write address was detected within the KEU address space.<br>0 No error detected<br>1 Address error |

Table 10-56. KEU Interrupt Status Register Signals Description (continued)

| Bits | Name | Description |
|---|---|---|
| 58 | OFE | Output FIFO error. The KEU output FIFO was non-empty upon write of the KEU data size register.<br>0 No error detected<br>1 Output FIFO non-empty error |
| 59 | IFE | Input FIFO error. The KEU input FIFO was non-empty upon generation of the done interrupt.<br>0 No error detected<br>1 Input FIFO non-empty error |
| 60 | — | Reserved |
| 61 | IFO | Input FIFO overflow. The KEU input FIFO has been pushed while full.<br>0 No error detected<br>1 Input FIFO has overflowed |
| 62 | OFU | Output FIFO underflow. The KEU output FIFO was read while empty.<br>0 No error detected<br>1 Output FIFO has underflow error |
| 63 | — | Reserved |

### 10.7.5.7 KEU Interrupt Mask Register (KEUIMR)

The KEUIMR controls the setting of bits in the KEU interrupt status register (KEUISR), as described in Section 10.7.5.6, “KEU Interrupt Status Register (KEUISR)”. If a KEUIMR bit is set, then the corresponding KEUISR bit is always zero.

Masking an error bit allows for a hardware error condition to go potentially undetected. Therefore, extreme care should be taken when masking errors, as invalid results may be produced. It is recommended that errors only be masked during debug operation. This register may be reset by resetting the KEU.

The KEUIMR fields are shown in Figure 10-70. The fields are defined in Table 10-57.

| | 0 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | — | ICE | — | IE | ERE | CE | KSE | DSE | DE | AE | OFE | IFE | — | IFO | OFU | — |
| Reset | 0 | 0 | 0 | 1 | 0 | | | | | | | | | | | |
| R/W | R/W | | | | | | | | | | | | | | | |
| Addr | KEU 0x3_E038 | | | | | | | | | | | | | | | |

Figure 10-70. KEU Interrupt Mask Register

Table 10-57 describes the KEU interrupt mask register fields.

**Table 10-57. KEU Interrupt Mask Register Fields Description**

| Bits | Name | Description |
|---|---|---|
| 0–48 | — | Reserved |
| 49 | ICE | Integrity check error.<br>0 ICV check error enabled. WARNING: Do not enable this EU status writeback (see bits IWSE and AWSE in Section 10.4.4.1, “Channel Configuration Register (CCR)” is used.<br>1 ICV check error disabled |
| 50 | — | Reserved |
| 51 | IE | Internal error. An internal processing error was detected while performing encryption.<br>0 Internal error enabled<br>1 Internal error disabled |
| 52 | ERE | Early read error. A KEU context or IV register was read while the KEU was performing encryption.<br>0 Early read error enabled<br>1 Early read error disabled |
| 53 | CE | Context error. A KEU key register, the key size register, data size register, mode register, or IV register was modified while the KEU was performing encryption.<br>0 Context error enabled<br>1 Context error disabled |
| 54 | KSE | Key size error. An inappropriate value (not 16 or 32 bytes) was written to the KEU key size register.<br>0 Key size error enabled<br>1 Key size error disabled |
| 55 | DSE | Data size error. Indicates that the number of bits to process is out of range.<br>0 Data size error enabled<br>1 Data size error disabled |
| 56 | DE | Data error. Indicates that invalid data was written to a register or a reserved mode bit was set.<br>0 Data error enabled<br>1 Data error disabled |
| 57 | AE | Address error. An illegal read or write address was detected within the KEU address space.<br>0 Address error enabled<br>1 Address error disabled |
| 58 | OFE | Output FIFO error. The KEU output FIFO was detected non-empty upon write of the KEU data size register.<br>0 Output FIFO non-empty error enabled<br>1 Output FIFO non-empty error disabled |
| 59 | IFE | Input FIFO error. The KEU input FIFO was detected non-empty upon generation of done interrupt.<br>0 Input FIFO non-empty error enabled<br>1 Input FIFO non-empty error disabled |
| 60 | — | Reserved |
| 61 | IFO | Input FIFO overflow. The KEU input FIFO was pushed while full.<br>0 Input FIFO overflow error enabled<br>1 Input FIFO overflow error disabled |

Table 10-57. KEU Interrupt Mask Register Fields Description (continued)

| Bits | Name | Description |
|---|---|---|
| 62 | OFU | Output FIFO underflow. The KEU output FIFO was read while empty.<br>0 Output FIFO underflow error enabled<br>1 Output FIFO underflow error disabled |
| 63 | — | Reserved |

### 10.7.5.8 KEU Data Out Register (f9 MAC) (KEUDOR)

Following a done interrupt, the read-only KEU data out register holds the f9 message authentication code. A 64-bit value is returned. This value may be truncated to 32 bits for some applications. Writing to this location results in an address error reflected in the KEU interrupt status register.

| | 0 63 |
|---|---|
| Field | KEU Data Out Register (f9 MAC) |
| Reset | 0x0000_0000_0000_0000 |
| R/W | R |
| Addr | KEU 0x3_E048 |

Figure 10-71. KEU Data Out Register (f9 MAC)

**NOTE**

According to the ETSI/SAGE 3GPP specification for f9 (version 1.2), only 32 bits of the final MAC are used. This corresponds to the lower 4 bytes of the KEU data out register.

### 10.7.5.9 KEU End of Message Register (KEUEMR)

The KEU end of message register, shown in Figure 10-72, is used to signal to the KEU that the final message block has been written to the input FIFO (in channel-driven access, this signaling is done automatically). The KEU will not process the last block of data in its input FIFO until this register is written. Once the end of message register is written, the KEU processes any remaining data in the input FIFO and generates the done interrupt.

The value written to this register does not matter. A read of this register always returns a zero value.

| | 0 63 |
|---|---|
| Field | — |
| Reset | 0 |
| R/W | R/W |
| Addr | KEU 0x3_E050 |

Figure 10-72. KEU End of Message Register

### 10.7.5.10 KEU IV_1 Register (KEUIV1)

The KEU IV_1 register is a general purpose IV register, shown in Figure 10-73, is used during the initialization phase of the f8 algorithms for 3GPP, GSM A5/3, EDGE A5/3, GPRS GEA3, and f9 algorithm for 3GPP. The appropriate value as defined by the standards for each algorithm must be written before a new message is started.

After the initialization phase has been completed, the KEU IV_1 register is no longer used for the remainder of f8 processing. However, if 3GPP f9 is selected because the KEU IV_1 register contains the direction bit as defined by the 3GPP standard, the KEU IV_1 register must be written back during context switches to complete the generation of the 3GPP MAC.

<table>
<tr><th></th><th>0 31</th><th>32 36</th><th>37</th><th>38 39</th><th>40 47</th><th>48 63</th></tr>
<tr><td>Field</td><td>CC</td><td>CB</td><td>CD</td><td>00</td><td>CA</td><td>CE</td></tr>
<tr><td>Reset</td><td colspan="6">0</td></tr>
<tr><td>R/W</td><td colspan="6">R/W</td></tr>
<tr><td>Addr</td><td colspan="6">KEU 0x3_E100</td></tr>
</table>

**Figure 10-73. KEU IV_1 Register**

Table 10-58 describes the KEU IV_1 register fields.

**Table 10-58. KEU IV_1 Register Fields Description**

<table>
<tr><th>Bits</th><th>Field</th><th>3GPP Definition</th><th>GSM A5/3 Definition</th><th>EDGE A5/3 Definition</th><th>GPRS GEA3 Definition</th></tr>
<tr><td>0–31</td><td>CC</td><td>Count</td><td>Count</td><td>0000000000 | | Count</td><td>Frame dependent input value (32-bits)</td></tr>
<tr><td>32–36</td><td>CB</td><td>Bearer</td><td>00000</td><td>00000</td><td>00000</td></tr>
<tr><td>37</td><td>CD</td><td>Direction bit</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>38–39</td><td>0</td><td>00</td><td>00</td><td>00</td><td>00</td></tr>
<tr><td>40–47</td><td>CA</td><td>00000000</td><td>00001111</td><td>11110000</td><td>11111111</td></tr>
<tr><td>48–63</td><td>CE</td><td>000000000000000</td><td>0000000000000000</td><td>0000000000000000</td><td>0000000000000000</td></tr>
</table>

The following figure shows how the KEU IV_1 register can be differentiated for different applications.

<table>
<tr><th></th><th>0 31</th><th>32 36</th><th>37</th><th>38 39</th><th>40 47</th><th>48 63</th></tr>
<tr><td>3GPP (f8)</td><td>Count</td><td>Bearer</td><td>Dir.</td><td>00</td><td>00000000</td><td>000000000000000</td></tr>
<tr><td>GSM (A5/3)</td><td>Count</td><td>00000</td><td>0</td><td>00</td><td>00001111</td><td>000000000000000</td></tr>
<tr><td>EDGE (A5/3)</td><td>0000000000 || Count</td><td>00000</td><td>0</td><td>00</td><td>11110000</td><td>000000000000000</td></tr>
<tr><td>GPRS (GEA3)</td><td>32 bit Frame Dependent Input Value</td><td>00000</td><td>0</td><td>00</td><td>11111111</td><td>000000000000000</td></tr>
</table>

**NOTE**

It is the responsibility of the user to ensure that fields of the KEU IV_1 register are programmed correctly in accordance with the algorithm selected.

### 10.7.5.11 KEU ICV_In Register (KEUICV)

If ICV checking is required, then the value to be compared with the computed f9 MAC value must be written to the KEU ICV_In register before data size is written. As the KEU ICV_In register is in between IV_1 and IV_2, any descriptor operation that loads IV_2 must also load ICV_In. If CICV = 0, the ICV_In register should be loaded with 0x0000_0000_0000_0000.

### 10.7.5.12 KEU IV_2 Register (FRESH) (KEUIV2)

The KEU IV_2 register, shown in Figure 10-74, holds the f9 value, FRESH, which is used during the initialization phase of the 3GPP f9 algorithm. This value is ignored when the f8 algorithm is selected. The FRESH value must be written to bits 0:31 of the KEU IV_2 register before a new message to be processed with 3GPP f9 is started. After the initialization phase has been completed, the KEU IV_2 register is no longer used during message processing. The KEU IV_2 register need not be written during context switches.

| | 0 … 31 | 32 … 63 |
|---|---|---|
| Field | FRESH | 00000000 |
| Reset | 0 | |
| R/W | R/W | |
| Addr | KEU 0x3_E110 | |

**Figure 10-74. KEU IV_2 Register (FRESH)**

### 10.7.5.13 KEU Context Data Registers (KEUC*n*)

The KEU includes six 64-bit KEU context data registers that store the running context used to process a message. The KEU context data registers must be read when changing context and are restored to their original values to resume processing of a partial message. For f8 and 3GPP f9 modes, all 64-bit KEU context data registers must be read to retrieve context, and all six registers must be written back to restore context. The context must be written prior to the key data. If the any of the KEU context data registers are written during message processing, a context error is generated. All KEU context data registers are cleared when a hard/soft reset or initialization is performed.

**NOTE**

For descriptor operation, if the entire context is unloaded for later reuse, the context data size must be 72 bytes, and the output consists of KEU IV_1, KEU ICV_In, KEU IV_2, and six KEU context data registers. For operations performing processing of partial messages, if the context is unloaded, the PE bit in the KEU mode register must not be set. Also, for partial message processing, if the context is reloaded, the INT bit in the KEU mode register must not be set.

### 10.7.5.14 KEU Key Data Registers_1 and _2 (Confidentiality Key) (KEUKD*n*)

The first two KEU key data registers, shown in Figure 10-76 and Figure 10-78, together hold one 128-bit key that is used for f8 encryption/decryption. The KEU key data register_1, (CK-high), holds the first 8 bytes (1–8). The KEU key data register_2, (CK-low), holds the second 8 bytes (9–16). The KEU key data registers must be written before message processing begins and cannot be written while the block is processing data, or else, a context error occurs. Reading from either of these registers causes an address error, which is reflected in the KEU interrupt status register.

Address KEU 0x3_E400 Access: Write-only

| | 0 … 63 |
|---|---|
| R | |
| W | KEU Key Data Register _1 (CK-high) |
| Reset | All zeros |

**Figure 10-75. KEU Key Data Register_1 (CK-high)**

**Figure 10-76. KEU Key Data Register_1 (CK-high)**

Address KEU 0x3_E408 Access: Write-only

| | 0 … 63 |
|---|---|
| R | |
| W | KEU Key Data Register _2 (CK-low) |
| Reset | All zeros |

**Figure 10-77. KEU Key Data Register_2 (CK-Low)**

**Figure 10-78. KEU Key Data Register_2 (CK-Low)**

### 10.7.5.15 KEU Key Data Registers _3 and _4 (Integrity Key) (KEUKD*n*)

The third and fourth KEU key data registers, shown in Figure 10-80 and Figure 10-82, together hold one 128-bit key that is used for f9 message authentication. The KEU key data register_3, (IK-high), holds the first 8 bytes (1–8). The KEU key data register_4, (IK-low), holds the second 8 bytes (9–16). The KEU key data registers must be written before message processing begins and cannot be written while the block is processing data, or else, a context error occurs.

If f9 only mode is set in the KEU mode register, the integrity key data may be optionally written to the KEU key data registers_1 and KEU key data registers_2. This eliminates the need for the host to offset from the base key address to write to the KEU key data registers_3 and KEU key data registers_4 while using the KEU exclusively for the f9 integrity function.

Reading from either of these registers causes an address error, which is reflected in the KEU interrupt status register.

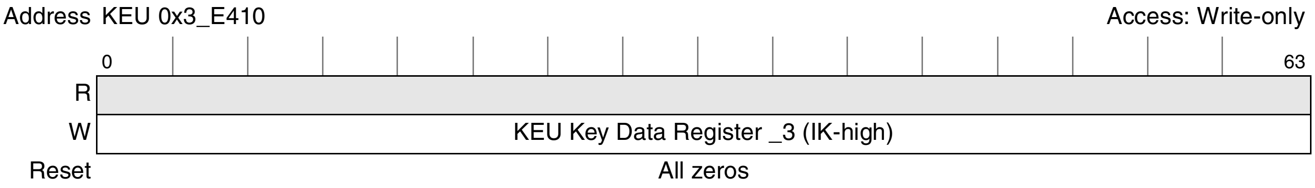


**Figure 10-79. KEU Key Data Register_3 (IK-high)**

**Figure 10-80. KEU Key Data Register_3 (IK-high)**

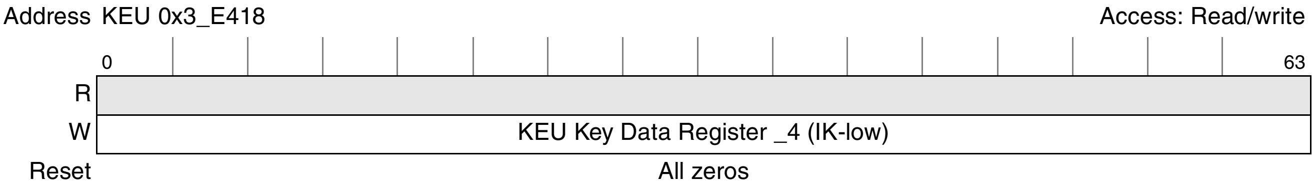


**Figure 10-81. KEU Key Data Register_4 (IK-low)**

**Figure 10-82. KEU Key Data Register_4 (IK-low)**

### 10.7.5.16 KEU FIFOs

KEU uses an input FIFO/output FIFO pair to hold data before and after the encryption process. Normally, the channels control all access to these FIFOs. For host-controlled operation, a write to anywhere in the KEU FIFO address space enqueues data to the KEU input FIFO, and a read from anywhere in the KEU FIFO address space de-queues data from the KEU output FIFO.

A write to the input FIFO go first to a staging register, which can be written by byte, word (4 bytes), or dword (8 bytes). When all 8 bytes of the staging register have been written, the entire dword is automatically enqueued into the FIFO. If any byte is written twice between enqueues, it causes an error interrupt of type AE from the EU. When writing the last portion of data, it is not necessary to write all 8 bytes. The last bytes remaining in the staging register are automatically padded with zeros and forced into the input FIFO when the KEU end of message register is written.

The output FIFO is readable by byte, word, or dword. When all 8 bytes of the head dword are read, the dword is automatically dequeued from the FIFO so that the next dword (if any) becomes available for reading. If any byte is read twice between de-queues, it causes an error interrupt of type AE from the EU.

Overflows and underflows caused by reading or writing the KEU FIFOs are reflected in the KEU interrupt status register.

The KEU fetches data 64 bits at a time from the KEU Input FIFO. During f8 processing, the input data is XORed with the generated keystream and the results are placed in the KEU output FIFO. During f9 processing, the input data is hashed with the integrity key and the resulting MAC is placed in the KEU data out register. The output size is the same as the input size.

## 10.7.6 Message Digest Execution Unit (MDEU)

This section contains details about the Message Digest Execution Unit (MDEU), including modes of operation, status and control registers, and FIFO.

Most of the registers described here would not normally be accessed by the host. They are documented here mainly for debug purposes. In typical operation, the MDEU is used through channel-controlled access, which means that most reads and writes of MDEU registers are directed by the SEC channels. Driver software would perform host-controlled register accesses only on a few registers for initial configuration and error handling.

### 10.7.6.1 ICV Checking in MDEU

This EU includes an ICV checking feature, that is, it can generate an ICV and compare it to another supplied ICV. The pass/fail result of this ICV check can be returned to the host either by interrupt by a writeback of EU status fields into host memory, but not by both methods at once.

To signal the ICV checking result by status writeback, turn on either the IWSE bit or AWSE bit in the Channel Configuration Register (see Section 10.4.4.1, “Channel Configuration Register (CCR)”), and mask the ICE bit in the interrupt mask register (Section 10.7.6.9, “MDEU Interrupt Mask Register”). In this case the normal done signaling (by interrupt or writeback) is undisturbed.

To signal the ICV checking result by interrupt, unmask the ICE bit in the interrupt mask register and turn off the IWSE and AWSE bits in the Channel Configuration Register. If there is no ICV mismatch, then the normal done signaling (by interrupt or writeback) occurs. When there is an ICV mismatch, there is an error interrupt to the host, but no channel done interrupt or writeback.

### 10.7.6.2 MDEU Mode Register

The MDEU Mode Register is used to program the function of the MDEU. In channel-driven access, bits 56-63 of this register are specified by the user through the MODE0 or MODE1 field of the descriptor header. The remaining two bits are supplied by the channel and thus are not under direct user control.

The two bits supplied by the channel are bits that control the meanings of other mode register fields. They are the MDEU_B bit, and the NEW bit.

The MDEU_B bit determines which of two sets of algorithms is available through the ALG bits. The two sets of algorithms are referred to as the MDEU-A set (MD5, SHA-1, SHA-224, and SHA-256) and the MDEU-B set (SHA-224, SHA-256, SHA-384, and SHA-512). MDEU_B = 0 selects the MDEU-A set, and MDEU_B = 1 selects the MDEU-B set. In channel-driven operation, the MDEU_B mode bit is supplied by the channel, based on the EU_SEL field of the descriptor header, where the user can choose MDEU-A or MDEU-B (see Table 10-6).

The NEW bit determines the configuration of other mode register fields as shown in Figure 10-83 and Figure 10-84. The “new” configuration (NEW=1) is used only by TLS/SSL descriptor types (1000_1, 1001_1). The old configuration (NEW=0) is used by all other descriptor types. The old configuration is the same as the one used in SEC 2.0, except for the CICV and SMAC bits. When MDEU is configured by the Polychannel, the value of NEW is determined by the descriptor type field of the descriptor header.

The mode register is cleared when the MDEU is reset or re-initialized. Setting a reserved mode bit generates a data error. If the mode register is modified during processing, a context error is generated.

| | 0 50 | 51 | 52 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 63 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | — | MDEU_B | — | NEW=0 | — | CONT | CICV | SMAC | INIT | HMAC | PD | ALG |
| Reset | 0 | | | | | | | | | | | |
| R/W | R/W | | | | | | | | | | | |
| Addr | MDEU 0x3_6000 | | | | | | | | | | | |

**Figure 10-83. MDEU Mode Register in Old Configuration**

Table 10-59 describes MDEU Mode Register fields in old configuration.

**Table 10-59. MDEU Mode Register in Old Configuration**

| Bits | Name | Description |
|---|---|---|
| The following bits are described for information only. They are not under direct user control. | | |
| 0–50 | — | Reserved |
| 51 | MDEU_B | Selects which algorithms are enabled by the ALG bits.<br>0 MDEU-A enables selection between SHA-1, SHA-256, MD5, and SHA-224<br>1 MDEU-B enables selection between SHA-384, SHA-256, SHA-512, and SHA-224. |
| 52-53 | — | Reserved, must be cleared. |
| 54 | NEW (=0) | Determines the configuration of the MDEU Mode Register. This table shows the configuration for NEW=0. |
| 55 | — | Reserved, must be cleared. |
| The following bits are controlled through the MODE0 or MODE1 fields of the descriptor header. | | |
| 56 | CONT | Continue: Most operations require this bit to be cleared. It is set only when the data to be hashed is spread across multiple descriptors.<br>The value programmed in PD must be opposite to the value in this bit.<br>0 Do autopadding and complete the message digest. Used when the entire hash is performed with one descriptor, or on the last of a sequence of descriptors.<br>1 This hash is continued in a subsequent descriptor. Do not autopad and do not complete the message digest. |
| 57 | CICV | Compare Integrity Check Values:<br>0 Normal operation; no ICV checking.<br>1 After the message digest (ICV) is computed, compare it to the data in the MDEU's input FIFO. If the ICVs do not match, send an error interrupt to the channel. The number of bytes to be compared is given by the ICV size register.<br>Only applicable to descriptor types that provide for reading an ICV in value. |
| 58 | SMAC | Specifies whether to perform an SSL-MAC operation:<br>0 Normal operation<br>1 Perform an SSL3.0 MAC operation. This requires a key and key length. If this is set then the HMAC bit should be 0. |

**Table 10-59. MDEU Mode Register in Old Configuration (continued)**

| Bits | Name | Description |
|---|---|---|
| 59 | INIT | Initialization Bit: Most operations require this bit to be set. Cleared only for operations that load context from a known intermediate hash value.<br>0 Do not initialize digest registers. In this case the registers must be loaded from a hash context pointer in the descriptor. When the data to be hashed is spread across multiple descriptors, this bit must be 0 on all but the first descriptor.<br>1 Do an algorithm-specific initialization of the digest registers. |
| 60 | HMAC | Specifies whether to perform an HMAC operation:<br>0 Normal operation<br>1 Perform an HMAC operation. This requires a key and key length. If this is set then the SMAC bit should be 0. |
| 61 | PD | This bit must be programmed opposite to the CONT bit. |
| 62-63 | ALG | Message Digest algorithm selection<br>00 if MDEU-B, then SHA-384. If MDEU-A, then SHA-160 algorithm (full name for SHA-1)<br>01 SHA-256 algorithm<br>10 if MDEU-B, then SHA-512. If MDEU-A, then MD5 algorithm<br>11 SHA-224 algorithm |

|  | 0 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | — |  | MDEU_B | — | STIB | NEW=1 | — | CONT | CICV | SMAC | INIT | HMAC | EALG | ALG |  |
| Reset | 0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| R/W | R/W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Addr | MDEU 0x3_6000 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

**Figure 10-84. MDEU Mode Register in New Configuration**

Table 10-60 describes MDEU Mode Register fields in new configuration.

**Table 10-60. MDEU Mode Register in New Configuration**

| Bits | Name | Description |
|---|---|---|
| The following bits are described for information only. They are not under direct user control. | | |
| 0–50 | — | Reserved |
| 51 | MDEU_B | Selects which algorithms are enabled by the ALG bits.<br>0 MDEU-A enables selection between SHA-1, SHA-256, MD5, and SHA-224<br>1 MDEU-B enables selection between SHA-384, SHA-256, SHA-512, and SHA-224. |
| 52 | — | Reserved, must be cleared. |
| 53 | STIB | SSL/TLS inbound, block cipher:<br>0 Normal operation.<br>1 Special operation only for SSL/TLS inbound, block cipher. Upon receiving end of message notification, the MDEU performs a calculation involving the last valid byte of data written into its input FIFO (which is Pad Length) to compute a final data size. The MDEU then processes the amount of data specified by this data size, and completes the message digest. |

Table 10-60. MDEU Mode Register in New Configuration (continued)

| Bits | Name | Description |
|---|---|---|
| 54 | NEW (=1) | Determines the configuration of the MDEU Mode Register. This table shows the configuration for NEW=1. |
| 55 | — | Reserved, must be cleared. |
| The following bits are controlled through the MODE0 or MODE1 fields of the descriptor header. | | |
| 56 | CONT | Continue: Most operations require this bit to be cleared. Set only when the data to be hashed is spread across multiple descriptors.<br>0 Do autopadding and complete the message digest. Used when the entire hash is performed with one descriptor, or on the last of a sequence of descriptors.<br>1 This hash is continued in a subsequent descriptor. Do not autopad and do not complete the message digest. |
| 57 | CICV | Compare Integrity Check Values:<br>0 Normal operation; no ICV checking.<br>1 After the message digest (ICV) is computed, compare it to the data in the MDEU's input FIFO. If the ICVs do not match, send an error interrupt to the channel. The number of bytes to be compared is given by the ICV size register. |
| 58 | SMAC | Specifies whether to perform an SSL-MAC operation:<br>0 Normal operation<br>1 Perform an SSL3.0 MAC operation. This requires a key and key length. If this is set then the HMAC bit should be 0. |
| 59 | INIT | Initialization Bit: Most operations require this bit to be set. Cleared only for operations that load context from a known intermediate hash value.<br>0 Do not initialize digest registers. In this case the registers must be loaded from a hash context pointer in the descriptor. When the data to be hashed is spread across multiple descriptors, this bit is set on all but the first descriptor.<br>1 Do an algorithm-specific initialization of the digest registers. |
| 60 | HMAC | Specifies whether to perform an HMAC operation:<br>0 Normal operation<br>1 Perform an HMAC operation. This requires a key and key length. If this is set then the SMAC bit should be 0. |
| 61 | EALG | The EALG (Extended Algorithm bit) and ALG (Algorithm) bits together specify the message digest algorithm, as follows:<br>000 if MDEU-B, then SHA-384. If MDEU-A, then SHA-160 algorithm (full name for SHA-1)<br>001 SHA-256 algorithm<br>010 if MDEU-B, then SHA-512. If MDEU-A, then MD5 algorithm<br>011 SHA-224 algorithm<br>others: Reserved |
| 62-63 | ALG | |

## 10.7.6.3 Recommended Settings for MDEU Mode Register

The most common task likely to be executed by the MDEU is HMAC generation. HMACs are used to provide message integrity within a number of security protocols, including IPsec, and TLS. The SSL 3.0 protocol uses a slightly different SSL-MAC. If an HMAC or SSL-MAC is to be performed using a single descriptor (with the MDEU acting as sole or secondary EU), the following mode register bit settings should be used:

Table 10-61. Mode Register—HMAC or SSL-MAC Generated by Single Descriptor

| Bits | Field | Value | |
|---|---|---|---|
| | | for HMAC | for SSL-MAC |
| 56 | CONT | 0 (off) | 0 (off) |
| 58 | SMAC | 0(on) | 1(on) |
| 59 | INIT | 1(on) | 1(on) |
| 60 | HMAC | 1(on) | 0(on) |

To generate an HMAC for a message that is spread across a sequence of descriptors, the following mode register bit settings should be used:

Table 10-62. Mode Register—HMAC Generated Across a Sequence of Descriptors

| Bits | Field | Value | | |
|---|---|---|---|---|
| | | First Descriptor | Middle Descriptor(s) | Final Descriptor |
| 56 | CONT | 1 (on) | 1 (on) | 0 (off) |
| 59 | INIT | 1 (on) | 0 (off) | 0 (off) |
| 60 | HMAC | 1 (on) | 0 (off) | 1 (on) |

All descriptors other than the final descriptor must output the intermediate message digest for the next descriptor to reload as MDEU context.

SSL-MAC operations cannot be spread across a sequence of descriptors.

Additional information on descriptors can be found in Section 10.3, “Descriptors.”

### 10.7.6.4 MDEU Key Size Register

Displayed in Figure 10-85, this value indicates the number of bytes of key memory that should be used in HMAC generation. MDEU supports at most one block of key. MDEU generates a key size error if the value written to this register exceeds 64 bytes for MD5, SHA-1, SHA-224, or SHA-256. If algorithms SHA-384 or SHA-512 are selected, then MDEU generates a key size error if the value written to this register exceeds 128 bytes.

| | 0 55 | 56 63 |
|---|---|---|
| Field | — | Key Size |
| Reset | 0 | |
| R/W | R/W | |
| Addr | MDEU 0x3_6008 | |

Figure 10-85. MDEU Key Size Register

### 10.7.6.5 MDEU Data Size Register

The MDEU Data Size Register, shown in Figure 10-86, specifies the number of bits of data to be processed.

The Data Size field is a 21-bit signed number. Values written to this register are added to the current register value. Multiple writes are allowed. The MDEU processes data when there is a positive value in this register and there is data available in the MDEU input FIFO. (Negative values can arise in inbound processing, when it is necessary to hold back data from the MDEU until the pad length has been decrypted.)

Since the MDEU does not support bit offsets, bits 61–63 must be written as 0 and are always read as zero. Furthermore, when the CONT bit of the MDEU mode register is set, the data size must be a multiple of the block size (512 bits for MD5, SHA-1, SHA-224 and SHA-256; 1024 bits for SHA-384 and SHA-512). Violating either of these conditions causes a data size error (DSE in the MDEU interrupt status register).

This register is cleared when the MDEU is reset or re-initialized. At the end of processing, its contents has been decremented down to zero (unless there is an error interrupt).

**NOTE**

> Writing to the data size register allows the MDEU to enter auto-start mode. Therefore, the required context registers must be written prior to writing the data size.

| | 0 – 42 | 43 – 63 |
|---|---|---|
| Field | — | Data Size |
| Reset | 0 | |
| R/W | R/W | |
| Addr | MDEU 0x3_6010 | |

Figure 10-86. MDEU Data Size Register

### 10.7.6.6 MDEU Reset Control Register

This register, shown in Figure 10-87, allows three levels reset of just the MDEU, as defined by the three self-clearing bits.

Offset 0x3_6018 Access: Read/Write

| | 0 – 60 | 61 | 62 | 63 |
|---|---|---|---|---|
| R<br>W | — | RI | MI | SR |
| Reset | All zeros | | | |

Figure 10-87. MDEU Reset Control Register

Table 10-63 describes MDEU reset control register fields.

**Table 10-63. MDEU Reset Control Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–60 | — | Reserved |
| 61 | RI | Reset Interrupt. Writing this bit active high causes MDEU interrupts signaling done and error to be reset. It further resets the state of the MDEU interrupt status register.<br>0 No reset<br>1 Reset interrupt logic |
| 62 | MI | Module initialization is nearly the same as software reset, except that the MDEU Interrupt mask register remains unchanged.<br>0 No reset<br>1 Reset most of MDEU |
| 63 | SR | Software reset is functionally equivalent to hardware reset (the RESET# pin), but only for the MDEU. All registers and internal state are returned to their defined reset state.<br>0 No reset<br>1 Full MDEU reset |

### 10.7.6.7 MDEU Status Register

This status register, as seen in Figure 10-88, reflects the state of the MDEU internal signals. The majority of these internal signals reflect the state of low-level MDEU functions, such as data padding, key padding, etc., and are not important to the user, however the user should be aware that reads of this register especially during processing are likely to return non-zero values for many bits between 0:57. The 4 signals shown are those which are most likely to be of interest to the user.

The MDEU status register is read only. Writing to this location results in address error being reflected in the MDEU interrupt status register.

Offset 0x3_6028 Access: Read only

| | 0–57 | 58 | 59–60 | 61 | 62 | 63 |
|---|---|---|---|---|---|---|
| R | — | HALT | ICCR | EI | DI | RD |
| W | | | | | | |
| Reset | All zeros | | | | | |

**Figure 10-88. MDEU Status Register**

Table 10-64 describes MDEU status register fields.

**Table 10-64. MDEU Status Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–57 | — | Reserved |
| 58 | HALT | Halt. Indicates that the MDEU has halted due to an error.<br>0 MDEU not halted<br>1 MDEU halted<br>**Note:** Because the error causing the MDEU to stop operating may be masked before reaching the interrupt status register, the MDEU interrupt status register is used to provide a second source of information regarding errors preventing normal operation. |
| 59-60 | ICCR | Integrity Check Comparison Result<br>00 No integrity check comparison was performed.<br>01 The integrity check comparison passed.<br>10 The integrity check comparison failed.<br>11 Reserved<br>**Note:** A “passed” or “failed” result is generated only if ICV checking is enabled. |
| 61 | EI | Error interrupt: This status bit reflects the state of the error interrupt signal, as sampled by the controller interrupt status register (Section 10.5.4.2.2, “Interrupt Status Register (ISR)”).<br>0 MDEU is not signaling error<br>1 MDEU is signaling error |
| 62 | DI | Done interrupt: This status bit reflects the state of the done interrupt signal, as sampled by the Controller interrupt status register (Section 10.5.4.2.2, “Interrupt Status Register (ISR)”).<br>0 MDEU is not signaling done<br>1 MDEU is signaling done |
| 63 | RD | Reset Done. This status bit, when high, indicates that MDEU has completed its reset sequence, as reflected in the signal sampled by the appropriate channel.<br>0 Reset in progress<br>1 Reset done<br>**Note:** Reset Done resets to 0, but has typically switched to 1 by the time a user checks the register, indicating the EU is ready for operation. |

### 10.7.6.8 MDEU Interrupt Status Register

The interrupt status register indicates which unmasked errors have occurred and have generated error interrupts to the channel. Each bit in this register can only be set if the corresponding bit of the MDEU interrupt mask register is zero (see Section 10.7.6.9, “MDEU Interrupt Mask Register”).

If the MDEU interrupt status register is non-zero, the MDEU halts and the MDEU error interrupt signal is asserted to the controller (see Section 10.5.4.2.2, “Interrupt Status Register (ISR)”). In addition, if the MDEU is being operated through channel-controlled access, then an interrupt signal is generated to the channel to which this EU is assigned. The EU error then appears in bit 55 of the Channel Status Register (see Table 10-15) and generates a channel error interrupt to the controller.

If the interrupt status register is written from the host, 1s in the value written are recorded in the interrupt status register if the corresponding bit is unmasked in the interrupt mask register. All other bits are cleared. This register can also be cleared by setting the RI bit of the MDEU reset control register.

The MDEU interrupt status register fields are shown in Figure 10-89, and described in Table 10-65.

Offset 0x3_6030 Access: Read/Write

| | 0 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 60 | 61 | 62 63 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | ICE | — | IE | ERE | CE | KSE | DSE | ME | AE | — | IFO | — |
| Reset | All zeros | | | | | | | | | | | | |

**Figure 10-89. MDEU Interrupt Status Register**

Table 10-65 describes MDEU interrupt status register fields.

**Table 10-65. MDEU Interrupt Status Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–48 | — | Reserved |
| 49 | ICE | Integrity Check Error:<br>0 No error detected<br>1 Integrity check error detected. An ICV check was performed and the supplied ICV did not match the one computed by the MDEU. |
| 50 | — | Reserved |
| 51 | IE | Internal Error. Indicates the MDEU has been locked up and requires a reset before use.<br>0 No internal error detected<br>1 Internal error detected<br>**Note:** This bit is asserted any time an enabled error condition occurs and can only be cleared by resetting the MDEU. |
| 52 | ERE | Early Read Error. The MDEU context was read before the MDEU completed the hashing operation.<br>0 No error detected<br>1 Early read error |
| 53 | CE | Context Error. The MDEU key register, key size register, or data size register was modified while MDEU was hashing.<br>0 No error detected<br>1 Context error |
| 54 | KSE | Key Size Error. Two possible causes:<br>• A value greater than permitted was written to the MDEU key size register (128 bytes for SHA-384 and SHA-512; 64 bytes otherwise)<br>• In either a HMAC or SMAC operation, key size was not written prior to writing data size or receiving an end of message command.<br>0 No error detected<br>1 Key size error |
| 55 | DSE | Data Size Error. A value not a multiple of 512 bits (1024 bits for SHA-384 and SHA-512) while the MDEU mode register CONT bit is high.<br>0 No error detected<br>1 Data size error |
| 56 | ME | Mode Error. Bit is set if any of these error conditions is detected:<br>• any reserved bit of the Mode register is set<br>• the ALG field of the Mode register contains an illegal value<br>0 No error detected<br>1 Mode error |

**Table 10-65. MDEU Interrupt Status Register Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 57 | AE | Address Error. An illegal read or write address was detected within the MDEU address space.<br>0 No error detected<br>1 Address Error |
| 58–60 | — | Reserved |
| 61 | IFO | Input FIFO Overflow. The MDEU Input FIFO was pushed while full.<br>0 No overflow detected<br>1 Input FIFO has overflowed<br>**Note:** When operated through channel-controlled access, the SEC implements flow control, and FIFO size is not a limit to data input size. When operated through host-controlled access, the MDEU cannot accept FIFO inputs larger than 256 bytes without overflowing. |
| 62–63 | — | Reserved |

## 10.7.6.9 MDEU Interrupt Mask Register

The MDEU interrupt mask register, shown in Figure 10-90, controls the result of detected errors. For a given error (as defined in Section 10.7.6.8, “MDEU Interrupt Status Register”), if the corresponding bit in this register is set, then the error is disabled; no error interrupt occurs and the interrupt status register is not updated to reflect the error. If the corresponding bit is not set, then upon detection of an error, the interrupt status register is updated to reflect the error, causing assertion of the error interrupt signal, and causing the module to halt processing.

Offset 0x3_6038 Access: Read/Write

| | 0 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 60 | 61 | 62 63 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | ICE | — | IE | ERE | CE | KSE | DSE | ME | AE | — | IFO | — |

**Figure 10-90. MDEU Interrupt Mask Register**

Table 10-65 describes MDEU interrupt status register fields.

**Table 10-66. MDEU Interrupt Mask Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–48 | — | Reserved |
| 49 | ICE | Integrity Check Error. The supplied ICV did not match the one computed by the MDEU.<br>0 Integrity check error enabled. WARNING: Do not enable this if using EU status writeback (see bits IWSE and AWSE in Section 10.4.4.1, “Channel Configuration Register (CCR)”).<br>1 Integrity check error disabled |
| 50 | — | Reserved |
| 51 | IE | Internal Error. An internal processing error was detected while performing hashing.<br>0 Internal error enabled<br>1 Internal error disabled |

Table 10-66. MDEU Interrupt Mask Register Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 52 | ERE | Early Read Error. The MDEU register was read while the MDEU was performing hashing.<br>0 Early read error enabled<br>1 Early read error disabled |
| 53 | CE | Context Error. The MDEU key register, the key size register, the data size register, or the mode register, was modified while the MDEU was performing hashing.<br>0 Context error enabled<br>1 Context error disabled |
| 54 | KSE | Key Size Error. A value outside the bounds was written to the MDEU key size register<br>0 Key size error enabled<br>1 Key size error disabled |
| 55 | DSE | Data Size Error. An inconsistent value was written to the MDEU data size register:<br>0 Data size error enabled<br>1 Data size error disabled |
| 56 | ME | Mode Error. An illegal value was detected in the mode register.<br>0 Mode error enabled<br>1 Mode error disabled |
| 57 | AE | Address Error. An illegal read or write address was detected within the MDEU address space.<br>0 Address error enabled<br>1 Address error disabled |
| 58–60 | — | Reserved |
| 61 | IFO | Input FIFO Overflow. The MDEU input FIFO was pushed while full.<br>0 Input FIFO overflow error enabled<br>1 Input FIFO overflow error disabled |
| 62–63 | — | Reserved |

## 10.7.6.10 MDEU ICV Size Register

The MDEU ICV Size Register, shown in Figure 10-91, specifies the number of bytes of the ICV result to be compared if the MDEU performs ICV checking (see Section 10.7.6.2, “MDEU Mode Register”). The data to be compared to the MDEU result is supplied to the MDEU through its input FIFO.

This register is cleared when the MDEU is reset or re-initialized.

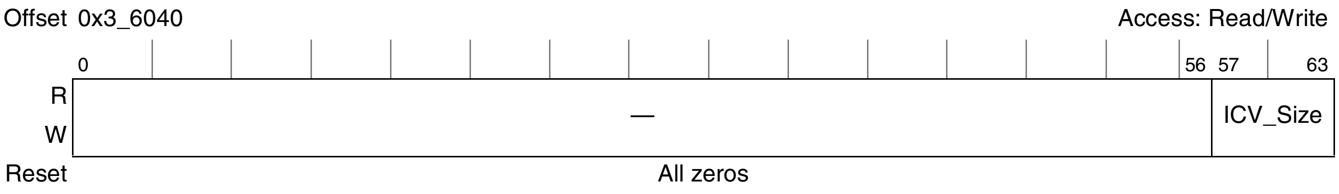


Figure 10-91. MDEU ICV Size Register

### 10.7.6.11 MDEU End of Message Register

The MDEU end of message register, shown in Figure 10-92, is used to signal to the MDEU that the final message block has been written to the input FIFO (in channel-driven access, this signaling is done automatically). The MDEU will not process the last block of data in its input FIFO until this register is written.

The value written to this register does not matter: ordinarily, zero is written. A read of this register always returns a zero value.

Offset 0x3_6050 Access: Write only

|  | 0 | 63 |
|---|---|---|
| R | — |  |
| W |  |  |
| Reset | All zeros |  |

**Figure 10-92. MDEU End of Message Register**

### 10.7.6.12 MDEU Context Registers

For MDEU, context consists of the hash plus the message length count. Write access to this register block allows continuation of a previous hash. Reading these registers provide the resulting message digest or HMAC, along with an aggregate bit count.

**NOTE**

All SHA algorithms are big endian. MD5 is little endian. The MDEU module internally reverses the byte order of the five registers A, B, C, D, and E upon writing to or reading from the MDEU context if the MDEU mode register indicates MD5 is the hash of choice. Most other endian considerations are performed as 8-byte swaps. In this case, 4-byte endianness swapping is performed within the A, B, C, D, and E fields as individual registers. Reading this memory location while the module is not done generates an error interrupt.

After a power on reset, all the MDEU context register values are cleared to 0. Figure 10-93 shows how the MDEU context registers are initialized if the INIT bit is set in the MDEU mode register. All registers are initialized, regardless of mode selected, however only the appropriate context register values are used in hash generation per the mode selected. The user typically doesn't care about the MDEU context register initialization values; they are documented for completeness in the event the user reads these registers using host-controlled access. MDEU reset through the MDEU reset control register (Figure 20-152) or SEC global software reset (Figure 20-37) does not clear these registers.

| | 0 31 | 32 63 | Register |
|---|---|---|---|
| **Algorithm** | | | Context offset 0x3_6100 |
| MD-5 | A = 0x01234567 | B = 0x89ABCDEF | |
| SHA-1 | A = 0x67452301 | B = 0xEFCDAB89 | |
| SHA-224 | A = 0xC1059ED8 | B = 0x367CD507 | |
| SHA-256 | A = 0x6A09E667 | B = 0xBB67AE85 | |
| SHA-384 | A = 0xcbbb9d5dc1059ed8 | | |
| SHA-512 | A = 0x6a09e667f3bcc908 | | |
| **Algorithm** | | | Context offset 0x3_6108 |
| MD-5 | C = 0xFEDCBA98 | D = 0x76543210 | |
| SHA-1 | C = 0x98BADCFE | D = 0x10325476 | |
| SHA-224 | C = 0x3070DD17 | D = 0xF70E5939 | |
| SHA-256 | C = 0x3C6EF372 | D = 0xA54FF53A | |
| SHA-384 | B = 0x629a292a367cd507 | | |
| SHA-512 | B = 0xbb67ae8584caa73b | | |
| **Algorithm** | | | Context offset 0x3_6110 |
| MD-5 | E = 0xF0E1D2C3 | F = 0x8C68059B | |
| SHA-1 | E = 0xC3D2E1F0 | F = 0x9B05688C | |
| SHA-224 | E = 0xFFC00B31 | F = 0x68581511 | |
| SHA-256 | E = 0x510E527F | F = 0x9B05688C | |
| SHA-384 | C = 0x9159015a3070dd17 | | |
| SHA-512 | C = 0x3c6ef372fe94f82b | | |

**Figure 10-93. MDEU Context Registers**

| | | | |
|---|---|---|---|
| **Algorithm** | | | Context offset 0x3_6118 |
| MD-5 | G = 0xABD9831F | H = 0x19CDE05B | |
| SHA-1 | G = 0x1F83D9AB | H = 0x5BE0CD19 | |
| SHA-224 | G = 0x64F98FA7 | H = 0xBEFA4FA4 | |
| SHA-256 | G = 0x1F83D9AB | H = 0x5BE0CD19 | |
| SHA-384 | D = 0xh152fecd8hf70e5939 | | |
| SHA-512 | D = 0xha54ff53ah5f1d36f1 | | |
| **Algorithm** | | | Context offset 0x3_6120 |
| MD5, SHA1, SHA-224, SHA-256 | Message Length Count = 0 | | |
| SHA-384 | E = 0x67332667ffc00b31 | | |
| SHA-512 | E = 0x510e527fade682d1 | | |
| **Algorithm** | | | Context offset 0x3_6128 |
| MD5, SHA1, SHA-224, SHA-256 | reserved | | |
| SHA-384 | F = 0x8eb44a8768581511 | | |
| SHA-512 | F = 0x9b05688c2b3e6c1f | | |
| **Algorithm** | | | Context offset 0x3_6130 |
| MD5, SHA1, SHA-224, SHA-256 | reserved | | |
| SHA-384 | G = 0xdb0c2e0d64f98fa7 | | |
| SHA-512 | G = 0x1f83d9abfb41bd6b | | |
| **Algorithm** | | | Context offset 0x3_6138 |
| MD5, SHA1, SHA-224, SHA-256 | reserved | | |
| SHA-384 | H = 0x47b5481dbefa4fa4 | | |
| SHA-512 | H = 0x5be0cd19137e2179 | | |
| **Algorithm** | | | Context offset 0x3_6140 |
| MD5, SHA1, SHA-224, SHA-256 | reserved | | |
| SHA-384, SHA-512 | Message Length Count = 0 | | |

**Figure 10-93. MDEU Context Registers (continued)**

If SHA-384 or SHA-512 are selected, then each of the registers A, B, C, D, E, F, G, H are 64-bits (instead of 32 bits for other hash algorithms). As a result, the base address for each context register is shifted to adjust.

#### 10.7.6.13 MDEU Key Registers

The MDEU maintains sixteen 64-bit registers for writing an HMAC key; only the first eight are used for MD5, SHA-1, SHA-224, or SHA-256. The IPAD and OPAD operations are performed automatically on the key data when required.

**NOTE**

All SHA algorithms are big endian. MD5 is little endian. The MDEU module internally reverses the endianness of the key upon writing to or reading from the MDEU key registers if the MDEU mode register indicates MD5 is the hash of choice.

#### 10.7.6.14 MDEU FIFOs

MDEU uses an input FIFO to hold data to be hashed (followed in some case by an ICV value for ICV checking). Normally, the channels control all access to this FIFO. For host-controlled operation, a write to anywhere in the MDEU FIFO address space enqueues data to the MDEU input FIFO, and a read from anywhere in this address space returns all zeros.

When the host writes to the MDEU FIFO (using host-controlled access), it can write to any FIFO address by byte, word (4 bytes), or dword (8 bytes). The MDEU assembles these bytes from left to right, so that the first bytes written are placed in the most significant bit-positions. Whenever the MDEU accumulates 8 bytes, this dword is automatically enqueued into the FIFO, and any remaining bytes are left-justified in preparation for assembling the next dword. It is not necessary to fill all bytes of the final dword. Any last bytes remaining in the staging register are automatically padded with zeros and forced into the input FIFO when the MDEU end of message register is written.

Overflows caused by writing the MDEU FIFO are reflected in the MDEU interrupt status register.

### 10.7.7 Public Key Execution Units (PKEU)

This section contains details about the public key execution unit (PKEU), including modes of operation, status and control registers, and parameter RAMs.

Most of the registers described here would not normally be accessed by the host. They are documented here mainly for debug purposes. In typical operation, the PKEU is used through channel-controlled access, which means that most reads and writes of PKEU registers are directed by the SEC channels. Driver software would perform host-controlled register accesses only on a few registers for initial configuration and error handling.

#### 10.7.7.1 PKEU Mode Register

This register specifies the internal PKEU routine to be executed. The mode register is cleared when the PKEU is reset or re-initialized. Setting a reserved mode bit generates a data error. If the mode register is modified during processing, a context error is generated.

Figure 10-94 shows the PKEU Mode Register, and Table 10-67 lists the possible values for the ROUTINE field.

Offset 0x3_C000 Access: Read/Write

| | 0 … 55 | 56 … 63 |
|---|---|---|
| R<br>W | — | ROUTINE |
| Reset | All zeros | |

**Figure 10-94. PKEU Mode Register**

For channel-controlled access to the PKEU, the descriptor type is determined by the ROUTINE to be used. The descriptor type used with each ROUTINE is listed in Table 10-67.

#### 10.7.7.2 PKEU Key Size Register

The key size register specifies the number of significant bytes to be used from PKEU Parameter Memory E in performing modular exponentiation or elliptic curve point multiplication. The range of values for this register, when performing either modular exponentiation or elliptic curve point multiplication, is from 1 to 512. Specifying a key size outside of this range causes a key size error (KSE) in the PKEU interrupt status register.

| | 0 … 51 | 52 … 63 |
|---|---|---|
| Field | — | Key Size |
| Reset | 0 | |
| R/W | R/W | |
| Addr | PKEU 0x3_C008 | |

**Figure 10-95. PKEU Key Size Register**

**Table 10-67. ROUTINE Field Description**

| Mode [56-63] | Routine Name | Routine Description | Descriptor Type |
|---|---|---|---|
| 0x00 | RESERVED | Reserved | NA |
| 0x01 | CLEARMEMORY | Clear memory | pkeu_mm |
| 0x02 | MOD_EXP | FP: Exponentiate mod N and deconvert from Montgomery format | pkeu_mm |
| 0x03 | MOD_R2MODN | FP: Compute Montgomery converter ($R^2$ mod N) | pkeu_mm |
| 0x04 | MOD_RRMODP | FP: Compute Montgomery converter for Chinese Remainder Theorem ($R_nR_p$ mod N) | pkeu_mm |
| 0x05 | EC_FP_AFF_PTMULT | FP EC: Multiply scalar times point in affine coordinates | pkeu_ptmul |

Table 10-67. ROUTINE Field Description (continued)

| Mode [56-63] | Routine Name | Routine Description | Descriptor Type |
|---|---|---|---|
| 0x06 | EC_F2M_AFF_PTMULT | F2m EC: Multiply scalar times point in affine coordinates | pkeu_ptmul |
| 0x07 | EC_FP_PROJ_PTMULT | FP EC: Multiply scalar times point in projective coordinates | pkeu_ptmul |
| 0x08 | EC_F2M_PROJ_PTMULT | F2m EC: Multiply scalar times point in projective coordinates | pkeu_ptmul |
| 0x09 | EC_FP_ADD | FP EC: Add two points in projective coordinates | pkeu_ptadd_dbl |
| 0x0A | EC_FP_DOUBLE | FP EC: Double a point in projective coordinates | pkeu_ptadd_dbl |
| 0x0B | EC_F2M_ADD | F2m EC: Add two points in projective coordinates | pkeu_ptadd_dbl |
| 0x0C | EC_F2M_DOUBLE | F2m EC: Double a point in projective coordinates | pkeu_ptadd_dbl |
| 0x0D | F2M_R2 | F2m: Compute Montgomery converter ($R^2$ mod N) | pkeu_mm |
| 0x0E | F2M_INV | F2m: Invert mod N | pkeu_mm |
| 0x0F | MOD_INV | FP: Invert mod N | pkeu_mm |
| 0x10 | MOD_ADD | FP: Add mod N | pkeu_mm |
| 0x20 | MOD_SUB | FP: Subtract mod N | pkeu_mm |
| 0x30 | MOD_MULT1_MONT | FP: Multiply mod N in Montgomery format | pkeu_mm |
| 0x40 | MOD_MULT2_DECONV | FP: Multiply mod N and deconvert from Montgomery format | pkeu_mm |
| 0x50 | F2M_ADD | F2m: Add mod N | pkeu_mm |
| 0x60 | F2M_MULT1_MONT | F2m: Multiply mod N in Montgomery format | pkeu_mm |
| 0x70 | F2M_MULT2_DECONV | F2m: Multiply mod N and deconvert from Montgomery format | pkeu_mm |
| 0x80 | RSA_SSTEP | FP: Exponentiate mod N (combines MOD_R2MODN, POLY_F2M_MULT1_MONT, and MOD_EXP) | pkeu_mm |
| 0x1d | MOD_EXP_TEQ | FP: Exponentiate mod N and deconvert from Montgomery format with timing equalization | pkeu_mm |
| 0x1e | RSA_SSTEP_TEQ | FP: Exponentiate mod N with timing equalization (combines MOD_R2MODN, EC_F2M_MULT1_MONT, and MOD_EXP_TEQ) | pkeu_mm |
| 0xFF | SPK_BUILD | Build PK data structure (data structure used by all elliptic curve routines) | pkeu_build |

### 10.7.7.3 PKEU AB Size Register

The AB size register (Figure 10-96) represents the size of each operand written into parameter memory A and parameter memory B in bits. An exact size in bits must be provided since a big- to little-endian re-alignment is performed based on this value. No error checking is performed as to whether the operand sizes are greater than the prime modulus or the field size written in N-ram. In other words, it is assumed that operands are modular reduced before being written into the PKEU module. This register must be written to before each write to parameter memory A or parameter memory B and must be written before each read of parameter memory A and parameter memory B if the amount of data being taken out is different than the amount of data put in A or B. The value written to the AB size register must also adhere to the constraints on parameters A and B, set by the chosen routine (see Table 10-67). The AB size register should not be read or written to while the PKEU is processing as this causes a ‘data modify during processing error.’ This register is cleared when the PKEU is reset. The maximum size acceptable is 4096 bits.

Offset 0x3_C040 Access: Read/Write

| | 0–51 | 52–63 |
|---|---|---|
| R<br>W | — | AB Size |
| Reset | All zeros | |

**Figure 10-96. PKEU AB Size Register**

### 10.7.7.4 PKEU Data Size Register

The PKEU Data Size Register, Figure 10-97, specifies the size (in bits) of the significant portion of the modulus or irreducible polynomial. The minimum size valid for all routines to operate properly is 33 bits. The maximum size to operate properly is 4096 bits. A value in bits larger than 4096 results in a data size error (see the DSE bit in Table 10-70).

Offset 0x3_C010 Access: Read/Write

| | 0–51 | 52–63 |
|---|---|---|
| R<br>W | — | Data Size |
| Reset | All zeros | |

**Figure 10-97. PKEU Data Size Register**

### 10.7.7.5 PKEU Reset Control Register

This register, Figure 10-98, contains three reset options specific to the PKEU.

| | 0–60 | 61 | 62 | 63 |
|---|---|---|---|---|
| Field | — | RI | MI | SR |
| Reset | 0 | | | |
| R/W | R/W | | | |
| Addr | PKEU 0x3_C018 | | | |

**Figure 10-98. PKEU Reset Control Register**

Table 10-68 describes the PKEU reset control register’s fields.

**Table 10-68. PKEU Reset Control Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–60 | — | Reserved |
| 61 | RI | Reset interrupt. Writing this bit active high causes PKEU interrupts signaling done and error to be reset. It further resets the state of the PKEU interrupt status register.<br>0 Do not reset<br>1 Reset interrupt logic |

**Table 10-68. PKEU Reset Control Register Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 62 | MI | Module initialization. Module initialization is nearly the same as Software Reset, except that the interrupt mask register remains unchanged. This module initialization includes execution of an initialization routine, completion of which is indicated by the RESET_DONE bit in the PKEU status register (Section 10.7.7.6, “PKEU Status Register”).<br>0 Do not reset<br>1 Reset most of PKEU |
| 63 | SR | SW reset. Software reset is functionally equivalent to hardware reset (the RESET# pin), but only for the PKEU. All registers and internal state are returned to their defined reset state. Upon negation of SW_RESET, the PKEU enters a routine to perform proper initialization of the parameter memories. The RESET_DONE bit in the PKEU status register indicates when this initialization routine is complete (Section 10.7.7.6, “PKEU Status Register”).<br>0 Do not reset<br>1 Full PKEU reset |
| 8–63 | — | Reserved |

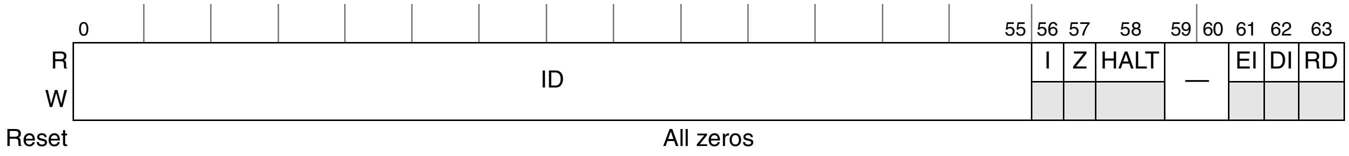
## 10.7.7.6 PKEU Status Register

This status register contains six fields which reflect the state of PKEU internal fields.

The PKEU status register is read only. Writing to this location results in address error being reflected in the PKEU interrupt status register.

Offset 0x3_C028

Access: Read only



| | 0–55 | 56 | 57 | 58 | 59–60 | 61 | 62 | 63 |
|---|---|---|---|---|---|---|---|---|
| R | ID | I | Z | HALT | — | EI | DI | RD |
| W | | | | | | | | |
| Reset | All zeros | | | | | | | |

**Figure 10-99. PKEU Status Register**

Table 10-69 describes the PKEU status register’s fields.

**Table 10-69. PKEU Status Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–55 | — | Reserved |
| 56 | I | Infinity. This bit reflects the state of the PKEU infinity detect bit when last sampled. Only particular instructions within routines cause infinity to be modified, so this bit should be used with great care. |
| 57 | Z | Zero. This bit reflects the state of the PKEU zero detect bit when last sampled. Only particular instructions within routines cause zero to be modified, so this bit should be used with great care. |

Table 10-69. PKEU Status Register Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 58 | HALT | Halt indicates that the PKEU has halted due to an error.<br>0 PKEU not halted<br>1 PKEU halted<br>**Note:** Because the error causing the PKEU to stop operating may be masked before reaching the interrupt status register, the PKEU interrupt status register is used to provide a second source of information regarding errors preventing normal operation. |
| 59-60 | — | Reserved |
| 61 | EI | Error interrupt: This status bit reflects the state of the error interrupt signal, as sampled by the controller interrupt status register (Section 10.5.4.2.2, “Interrupt Status Register (ISR)”).<br>0 PKEU is not signaling error<br>1 PKEU is signaling error |
| 62 | DI | Done interrupt: This status bit reflects the state of the done interrupt signal, as sampled by the Controller interrupt status register (Section 10.5.4.2.2, “Interrupt Status Register (ISR)”).<br>0 PKEU is not signaling done<br>1 PKEU is signaling done |
| 63 | RD | Reset Done. This status bit, when high, indicates that PKEU has completed its reset sequence, as reflected in the signal sampled by the appropriate channel.<br>0 Reset in progress<br>1 Reset done<br>**Note:** Reset Done resets to 0, but has typically switched to 1 by the time a user checks the register, indicating the EU is ready for operation. |

### 10.7.7.7 PKEU Interrupt Status Register

The interrupt status register indicates which unmasked errors have occurred and have generated error interrupts to the channel. Each bit in this register can only be set if the corresponding bit of the PKEU interrupt mask register is zero (see Section 10.7.7.8, “PKEU Interrupt Mask Register”).

If the PKEU interrupt status register is non-zero, the PKEU halts and the PKEU error interrupt signal is asserted to the controller (see Section 10.5.4.2.2, “Interrupt Status Register (ISR)”). In addition, if the PKEU is being operated through channel-controlled access, then an interrupt signal is generated to the channel to which this EU is assigned. The EU error then appears in bit 55 of the channel status register (see Table 10-15) and generates a channel error interrupt to the controller.

If the interrupt status register is written from the host, 1s in the value written are recorded in the interrupt status register if the corresponding bit is unmasked in the interrupt mask register. All other bits are cleared. This register can also be cleared by setting the RI bit of the PKEU reset control register.

The fields of the PKEU interrupt status register are shown in Figure 10-100, and described in Table 10-70.

Offset 0x3_C030 Access: Read/Write

| | 0 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 63 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | ID | | EVM | INV | EI | BE | CE | KSE | DSE | ME | AE | — | |
| Reset | All zeros | | | | | | | | | | | | |

**Figure 10-100. PKEU Interrupt Status Register**

Table 10-70 describes PKEU interrupt status register signals.

**Table 10-70. PKEU Interrupt Status Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0-48 | — | Reserved |
| 49 | EVM | Even modulus error. Indicates that an even modulus was supplied for a PK operation that requires an odd modulus.<br>0 No even modulus error detected<br>1 Even modulus error detected |
| 50 | INV | Inversion error. Indicates that the inversion routine has a zero operand.<br>0 No inversion error detected<br>1 Inversion error detected |
| 51 | IE | Internal error. An internal processing error was detected while the PKEU was operating.<br>0 No error detected<br>1 Internal error<br>**Note:** This bit is asserted any time an enabled error condition occurs and can only be cleared by setting the corresponding bit in the interrupt mask register or by resetting the PKEU. |
| 52 | BE | Boot Error. Indicates that either at boot (reset) sequence, RAM locations are not reset correctly.<br>0 No boot error detected<br>1 Boot error |
| 53 | CE | Context error. A PKEU key register, the key size register, the data size register, or mode register was modified while the PKEU was operating.<br>0 No error detected<br>1 Context error |
| 54 | KSE | Key size error. Value outside the bounds of 1–256 bytes was written to the PKEU key size register<br>0 No error detected<br>1 Key size error detected |
| 55 | DSE | Data size error. Value outside the bounds 97– 4096 bits was written to the PKEU data size register<br>0 No error detected<br>1 Data size error detected |
| 56 | ME | Mode error. An illegal value was detected in the mode register.<br>0 No error detected<br>1 Mode error<br>**Note:** Writing to reserved bits in a mode register is a likely source of error. |

Table 10-70. PKEU Interrupt Status Register Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 57 | AE | Address error. Illegal read or write address was detected within the PKEU address space.<br>0 No error detected<br>1 Address error |
| 58-63 | — | Reserved |

### 10.7.7.8 PKEU Interrupt Mask Register

The PKEU interrupt mask register (shown in Figure 10-101) controls the result of detected errors. For a given error (as defined in Section 10.7.7.7, “PKEU Interrupt Status Register”), if the corresponding bit in this register is set, then the error is disabled; no error interrupt occurs and the interrupt status register is not updated to reflect the error. If the corresponding bit is not set, then upon detection of an error, the PKEU interrupt status register is updated to reflect the error, causing assertion of the error interrupt signal, and causing the module to halt processing.

Offset 0x3_C038 Access: Read/Write

| | 0 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 63 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| R | ID | EVM | INV | EI | BE | CE | KSE | DSE | ME | AE | — |
| W | | | | | | | | | | | |

Figure 10-101. PKEU Interrupt Mask Register

Table 10-71 describes the PKEU interrupt mask register fields.

Table 10-71. PKEU Interrupt Mask Register Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0-48 | — | Reserved |
| 49 | EVM | Even modulus error<br>0 Even modulus error enabled<br>1 Even modulus error disabled |
| 50 | INV | Inversion error<br>0 Inversion error enabled<br>1 Inversion error disabled |
| 51 | IE | Internal error<br>0 Internal error enabled<br>1 Internal error disabled |
| 52 | BE | Boot error<br>0 Boot error enabled<br>1 Boot error disabled |
| 53 | CE | Context error<br>0 Context error enabled<br>1 Context error disabled |

Table 10-71. PKEU Interrupt Mask Register Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 54 | KSE | Key size error<br>0 Key size error enabled<br>1 Key size error disabled |
| 55 | DSE | Data size error<br>0 Data size error enabled<br>1 Data size error disabled |
| 56 | ME | Mode error<br>0 Mode error enabled<br>1 Mode error disabled |
| 57 | AE | Address error<br>0 Address error enabled<br>1 Address error disabled |
| 58-63 | — | Reserved |

### 10.7.7.9 PKEU End of Message Register

The PKEU end of message register in the PKEU is used to indicate the start of a new computation. Writing to this register causes the PKEU to execute the function requested by the ROUTINE field, according to the contents of the parameter memories described in Section 10.7.7.10, “PKEU Parameter Memories”.

The value written to this register does not matter: ordinarily, all zeros are written. A read of this register always returns a zero value.

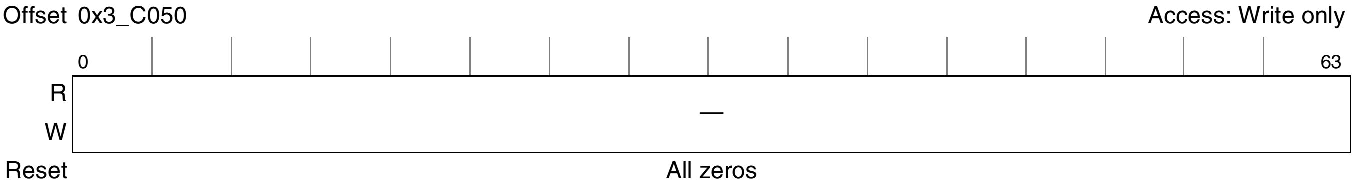


Figure 10-102. PKEU End of Message Register

### 10.7.7.10 PKEU Parameter Memories

The PKEU uses four 4096-bit memories to receive and store operands for the arithmetic operations the PKEU is asked to perform. In addition, results are stored in one particular parameter memory.

Data addressing within these memories is big-endian, that is, the most significant byte is stored in the lowest address.

#### 10.7.7.10.1 PKEU Parameter Memory A

This 4096 bit memory is used typically as an input parameter memory space. For modular arithmetic routines, this memory operates as one of the operands of the desired function. For elliptic curve routines, this memory is segmented into four 1024 bit memories, and is used to specify particular curve parameters and input values.

#### 10.7.7.10.2 PKEU Parameter Memory B

This 4096-bit memory is used typically as an input parameter memory space, as well as the result memory space. For modular arithmetic routines, this memory operates as one of the operands of the desired function, as well as the result memory space. For elliptic curve routines, this memory is segmented in to four 1024 bit memories, and is used to specify particular curve parameters and input values, as well as to store result values.

#### 10.7.7.10.3 PKEU Parameter Memory E

This 4096-bit memory is non-segmentable, and specifies the exponent for modular exponentiation, or the multiplier k for elliptic curve point multiplication. This memory space is write only; a read of this memory space causes address error to be reflected in the PKEU interrupt status register.

#### 10.7.7.10.4 PKEU Parameter Memory N

This 4096-bit memory is non-segmentable, and specifies the modulus for modular arithmetic and $F_p$ elliptic curve routines. For $F_2$m elliptic curve routines, this memory specifies the irreducible polynomial.

### 10.7.8 Random Number Generator Unit (RNGU)

This section contains details about the random number generator unit, including modes of operation, status and control registers, and FIFO.

The RNGU is an execution unit capable of generating 64-bit random numbers. It contains a True Random Number Generator (TRNG).The RNGU is designed to comply with the FIPS-140 standard for randomness and non-determinism.

The RNGU consists of five major functional blocks:

- 64-bit internal bus interface, registers, and FIFO
- True Random Number Generator (ring oscillator, LFSRs, Statistical Checker)
- Xseed Generator
- Pseudo-Random Number Generator (XKEY, SHA-1, FSM)
- Simultaneous Reseed LFSR

The states of the LFSRs in the TRNG are advanced at an unknown frequency determined by the ring oscillator clock. The entropy generated by this structure is then added into the XKEY structure of the PRNG during seed generation. Seed generation takes approximately 2,000,000 cycles as 20,000 bits of entropy are sampled from the output of the LFSRs of the TRNG.

After the initial seeding, the RNGU turns off the TRNG and uses solely the PRNG to generate random data. After 1,000,000 times through the algorithm the RNGU is once again seeded. This second seed occurs the next time through the algorithm by using data from the Simultaneous Reseed LFSR to modify the algorithm. The data in the simultaneous reseed LFSR comes directly from the TRNG as well and was being generated during the first 20,000 times through the PRNG algorithm after the initial seed was completed.

Most of the registers described here would not normally be accessed by the host. They are documented here mainly for debug purposes. In typical operation, the RNGU is used through channel-controlled access, which means that most reads and writes of RNGU registers are directed by the SEC channels. Driver software would perform host-controlled register accesses only on a few registers for initial configuration and error handling.

### 10.7.8.1 RNGU Mode Register

The RNGU Mode Register is a writable location but all mode bits are currently reserved. It is documented for the sake of consistency with the other EUs. The RNGU mode register is shown in Figure 10-103.

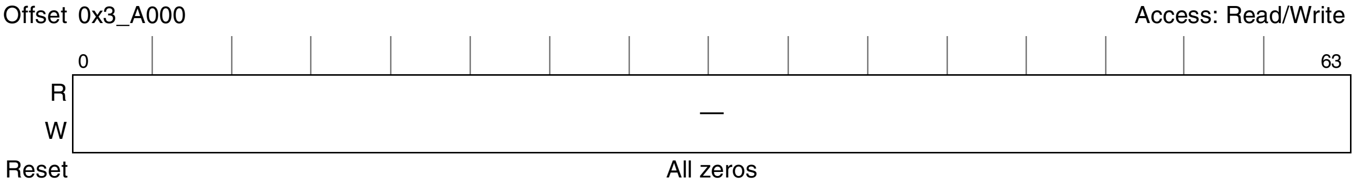


Figure 10-103. RNGU Mode Register

### 10.7.8.2 RNGU Data Size Register

The RNGU data size register is used to tell the RNGU to begin generating random data. The actual contents of the data size register does not affect the operation of the RNGU. After a reset and prior to the first write of data size, the RNGU builds entropy without pushing data onto the FIFO. Once the data size register is written, the RNGU begins pushing data onto the FIFO. One dword (64 bits) of data is pushed onto the FIFO every 112 cycles until the FIFO is full. The RNGU then attempts to keep the FIFO full.

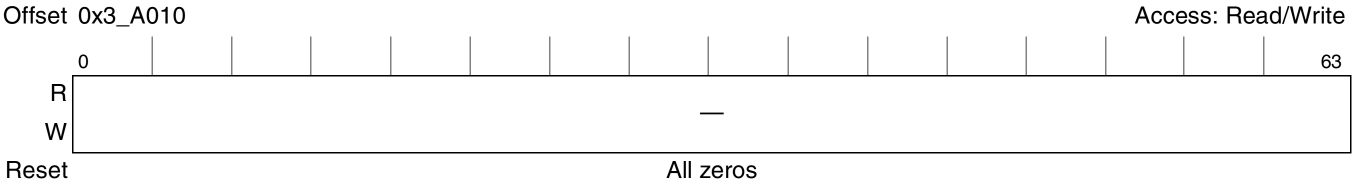


Figure 10-104. RNGU Data Size Register

### 10.7.8.3 RNGU Reset Control Register

This register, shown in Figure 10-105, contains three reset options specific to the RNGU.

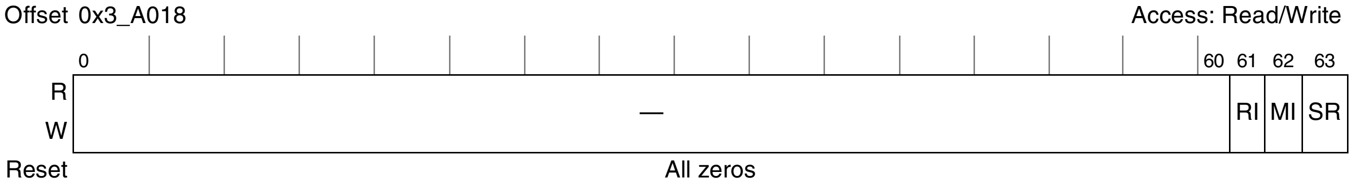


Figure 10-105. RNGU Reset Control Register

Table 10-72 describes RNGU reset control register fields.

**Table 10-72. RNGU Reset Control Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0-60 | — | Reserved |
| 61 | RI | Reset Interrupt. Writing this bit active high causes RNGU interrupts signaling done and error to be reset. It further resets the state of the RNGU interrupt status register.<br>0 No reset<br>1 Reset interrupt logic |
| 62 | MI | Module Initialization. This reset value performs enough of a reset to prepare the RNGU for another request, without forcing the internal control machines and the output FIFO to be reset, thereby invalidating stored random numbers or requiring reinvocation of a warm-up period. Module initialization is nearly the same as software reset, except that the interrupt mask register remains unchanged.<br>0 No reset<br>1 Reset most of RNGU |
| 63 | SR | Software reset is functionally equivalent to hardware reset (the RESET# pin), but only for the RNGU. All registers and internal state are returned to their defined reset state.<br>0 No reset<br>1 Full RNGU reset |
| 8–63 | — | Reserved |

## 10.7.8.4 RNGU Status Register

This RNGU status register, Figure 10-106, contains 6 fields that reflect the state of the RNGU internal signals.

The RNGU status register is read only. Writing to this location results in an address error being reflected in the RNGU interrupt status register.

Offset 0x3_A028 Access: Read only

| | 0–39 | 40–47 | 48–57 | 58 | 59–60 | 61 | 62 | 63 |
|---|---|---|---|---|---|---|---|---|
| R | — | OFL | — | HALT | — | EI | DI | RD |
| W | | | | | | | | |
| Reset | All zeros | | | | | | | |

**Figure 10-106. RNGU Status Register**

Table 10-73 describes RNGU status register fields.

**Table 10-73. RNGU Status Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–39 | — | Reserved |
| 40–47 | OFL | The number of dwords currently in the output FIFO |
| 48–57 | — | Reserved |

Table 10-73. RNGU Status Register Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 58 | HALT | Halt. Indicates that the RNGU has halted due to an error.<br>0 RNGU not halted<br>1 RNGU halted<br>Note: Because the error causing the RNGU to stop operating may be masked before reaching the interrupt status register, the RNGU interrupt status register is used to provide a second source of information regarding errors preventing normal operation. |
| 59–60 | — | Reserved |
| 61 | EI | Error interrupt: This status bit reflects the state of the error interrupt signal, as sampled by the Controller interrupt status register (Section 10.5.4.2.2, “Interrupt Status Register (ISR)”).<br>0 RNGU is not signaling error<br>1 RNGU is signaling error |
| 62 | DI | Done interrupt: This status bit reflects the state of the done interrupt signal, as sampled by the controller interrupt status register (Section 10.5.4.2.2, “Interrupt Status Register (ISR)”).<br>0 RNGU is not signaling done<br>1 RNGU is signaling done |
| 63 | RD | Reset Done. This status bit, when high, indicates that the RNGU has completed its reset sequence.<br>0 Reset in progress<br>1 Reset done<br>**Note:** Reset Done resets to 0, but has typically switched to 1 by the time a user checks the register, indicating the EU is ready for operation. |

### 10.7.8.5 RNGU Interrupt Status Register

The RNGU interrupt status register indicates which unmasked errors have occurred and have generated error interrupts to the channel. Each bit in this register can only be set if the corresponding bit of the RNGU interrupt mask register is zero (see Section 10.7.8.6, “RNGU Interrupt Mask Register”).

If the RNGU interrupt status register is non-zero, the RNGU halts and the RNGU error interrupt signal is asserted to the controller (see Section 10.5.4.2.2, “Interrupt Status Register (ISR)”). In addition, if the RNGU is being operated through channel-controlled access, then an interrupt signal is generated to the channel to which this EU is assigned. The EU error then appears in bit 55 of the Channel Pointer Status Register (see Table 10-15) and generates a channel error interrupt to the controller.

If the interrupt status register is written from the host, 1s in the value written are recorded in the interrupt status register if the corresponding bit is unmasked in the interrupt mask register. All other bits are cleared. This register can also be cleared by setting the RI bit of the RNGU Reset Control Register.

The bit fields of the RNGU interrupt status register are shown in Figure 10-107.

Offset 0x3_A030 Access: Read/Write

| | 0–50 | 51 | 52–55 | 56 | 57 | 58–61 | 62 | 63 |
|---|---|---|---|---|---|---|---|---|
| R<br>W | — | IE | — | ME | AE | — | OFU | — |
| Reset | All zeros | | | | | | | |

**Figure 10-107. RNGU Interrupt Status Register**

Table 10-74 describes the RNGU interrupt status register fields.

**Table 10-74. RNGU Interrupt Status Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–50 | — | Reserved |
| 51 | IE | Internal Error<br>0 No internal error detected<br>1 Internal error |
| 52–55 | — | Reserved |
| 56 | ME | Mode Error. Indicates that the host has attempted to write an illegal value to the mode register<br>0 Valid data<br>1 Invalid data error |
| 57 | AE | Address Error. An illegal read or write address was detected within the RNGU address space.<br>0 No error detected<br>1 Address error |
| 58–61 | — | Reserved |
| 62 | OFU | Output FIFO Underflow. The RNGU Output FIFO was read while empty.<br>0 No overflow detected<br>1 Output FIFO has underflowed |
| 63 | — | Reserved |

### 10.7.8.6 RNGU Interrupt Mask Register

The RNGU interrupt mask register controls the result of detected errors. For a given error (as defined in Section 10.7.8.5, “RNGU Interrupt Status Register”), if the corresponding bit in this register is set, then the error is disabled; no error interrupt occurs and the interrupt status register is not updated to reflect the error. If the corresponding bit is not set, then upon detection of an error, the interrupt status register is updated to reflect the error, causing assertion of the error interrupt signal, and causing the module to halt processing.

The bit fields of the RNGU interrupt mask register are shown in Figure 10-108.

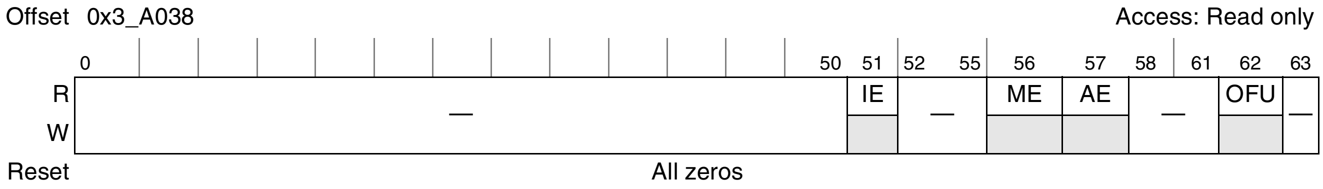


**Figure 10-108. RNGU Interrupt Mask Register**

Table 10-75 describes RNGU interrupt status register fields.

**Table 10-75. RNGU Interrupt Mask Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–50 | — | Reserved |
| 51 | IE | Internal Error. An internal processing error was detected while generating random numbers. This error is no longer maskable and can only be cleared by setting one of the reset bits in the Reset Control Register<br>0 Internal error enabled<br>1 Internal error disabled |
| 52–55 | — | Reserved |
| 56 | ME | Mode Error. An illegal value was detected in the mode register.<br>0 Mode error enabled<br>1 Mode error disabled |
| 57 | AE | Address Error. An illegal read or write address was detected within the RNGU address space.<br>0 Address error enabled<br>1 Address error disabled |
| 58–61 | — | Reserved |
| 62 | OFU | Output FIFO Underflow. RNGU Output FIFO was read while empty.<br>0 Output FIFO underflow error enabled<br>1 Output FIFO underflow error disabled |
| 63 | — | Reserved |

### 10.7.8.7 RNGU End of Message Register

The RNGU end of message register (shown in Figure 20-192) is a write-only register that can be used to start the RNGU. A write of any value to this register causes the RNGU to begin to produce random numbers in the FIFO.

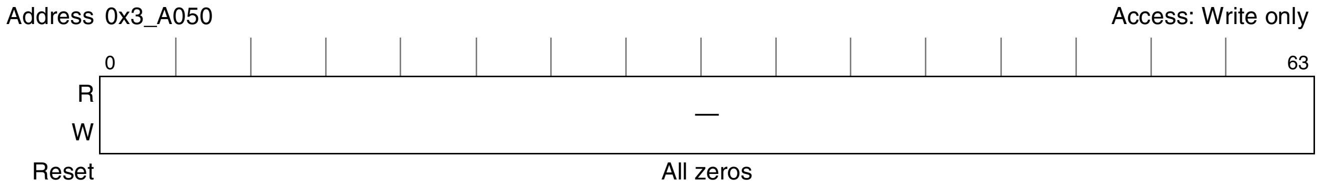


**Figure 10-109. RNGU End of Message Register**

### 10.7.8.8 RNGU Entropy Registers

RNGU allows the user to input entropy bits into the PRNG algorithm to modify the randomness of the RNGU. This group of registers are write-only, and all writes to these registers are ignored when the RNGU is busy. However when the RNGU is idle (FIFO is full or RNGU has not yet been started), all data written to these registers is used to modify the internal XKEY structure. These 64-bit registers cannot be written back-to-back—there must be a clock cycle in between writes, since the RNGU only processes 32 bits per cycle.

### 10.7.8.9 RNGU FIFO

RNGU uses an output FIFO to collect periodically sampled random 64-bit-words, with the intent that random data always be available for reading. Normally, the channels control all access to this FIFO. For host-controlled operation, a read from anywhere in the RNGU FIFO address space dequeues data from the RNGU output FIFO.

The output FIFO is readable by byte, word, or dword. When all 8 bytes of the head dword have been read, that dword is automatically dequeued from the FIFO so that the next dword (if any) becomes available for reading. If any byte is read twice between dequeues, it causes an error interrupt of type AE from the EU.

Underflows caused by reading or writing the RNGU output FIFO are reflected in the RNGU interrupt status register. Also, a write to the RNGU output FIFO space is reflected as an addressing error in the RNGU interrupt status register.

**NOTE**

Host reads of the RNGB FIFO should be performed on an 8-byte basis, regardless of how many bits of random number is actually required. Partial host reads can leave the RNGB FIFO in a state that results in a channel error.

# Chapter 11
# $I^2C$ Interfaces

This chapter describes the two inter-IC (IIC or $I^2C$) bus interfaces implemented on this device.

## 11.1 Introduction

The $I^2C$ bus is a two-wire—serial data (SDA) and serial clock (SCL)— bidirectional serial bus that provides a simple efficient method of data exchange between this device and other devices, such as microcontrollers, EEPROMs, real-time clock devices, A/D converters, and LCDs. Figure 11-1 shows a block diagram of the two $I^2C$ interfaces.

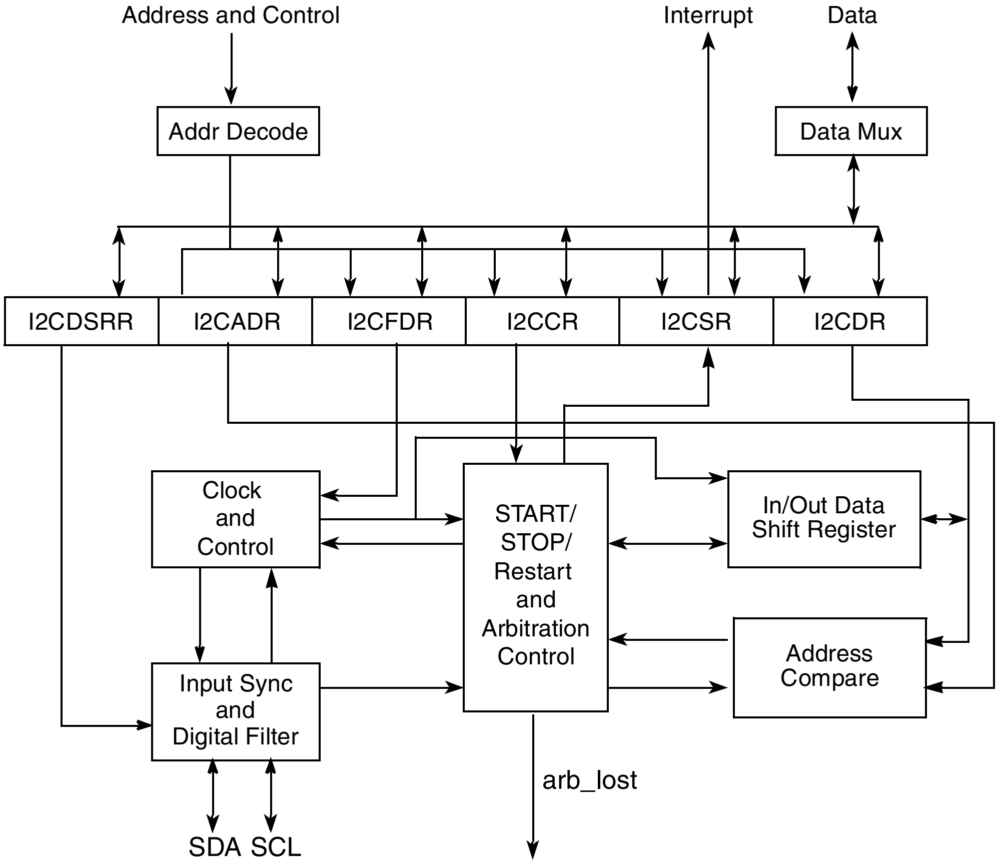


**Figure 11-1. $I^2C$ Block Diagram**

### 11.1.1 Overview

The two-wire I$^2$C bus minimizes interconnections between devices. The synchronous, multiple-master I$^2$C bus allows the connection of additional devices to the bus for expansion and system development. The bus includes collision detection and arbitration that prevent data corruption if two or more masters attempt to control the bus simultaneously.

### 11.1.2 Features

The I$^2$C interface includes the following features:

- Two-wire interface
- Multiple-master operation
- Arbitration lost interrupt with automatic mode switching from master to slave
- Calling address identification interrupt
- START and STOP signal generation/detection
- Acknowledge bit generation/detection
- Bus busy detection
- Software-programmable clock frequency
- Software-selectable acknowledge bit
- On-chip filtering for spikes on the bus

### 11.1.3 Modes of Operation

The I$^2$C units on this device can operate in one of the following modes:

- Master mode—The I$^2$C is the driver of the SDA line. It cannot use its own slave address as a calling address. The I$^2$C cannot be a master and a slave simultaneously.
- Slave mode—The I$^2$C is not the driver of the SDA line. The module must be enabled before a START condition from a non-I$^2$C master is detected.
- Interrupt-driven byte-to-byte data transfer—When successful slave addressing is achieved (and SCL returns to zero), the data transfer can proceed on a byte-to-byte basis in the direction specified by the R/$\overline{\text{W}}$ bit sent by the calling master. Each byte of data must be followed by an acknowledge bit, which is signaled from the receiving device. Several bytes can be transferred during a data transfer session.
- Boot sequencer mode—This mode can be used to initialize the configuration registers in the device after the I$^2$C1 module is initialized. Note that the device powers up with boot sequencer mode disabled as a default, but this mode can be selected with the cfg_boot_seq[0:1] power-on reset (POR) configuration signals that are located on the LGPL3 and LGPL5 signals.

Additionally, the following three I$^2$C-specific states are defined for the I$^2$C interface:

- START condition—This condition denotes the beginning of a new data transfer (each data transfer contains several bytes of data) and awakens all slaves.
- Repeated START condition—A START condition that is generated without a STOP condition to terminate the previous transfer.
- STOP condition—The master can terminate the transfer by generating a STOP condition to free the bus.

## 11.2 External Signal Descriptions

The following sections give an overview of signals and provide detailed signal descriptions.

### 11.2.1 Signal Overview

The I$^2$C interface uses the SDA and SCL signals, described in Table 11-1, for data transfer. Note that the signal patterns driven on SDA represent address, data, or read/write information at different stages of the protocol.

**Table 11-1. I$^2$C Interface Signal Descriptions**

| Signal Name | Idle State | I/O | State Meaning |
|---|---|---|---|
| Serial Clock (IIC*n*_SCL) | HIGH | I | When the I$^2$C module is idle or acts as a slave, SCL defaults as an input. The unit uses SCL to synchronize incoming data on SDA. The bus is assumed to be busy when SCL is detected low. |
| | | O | As a master, the I$^2$C module drives SCL along with SDA when transmitting. As a slave, the I$^2$C module drives SCL low for data pacing. |
| Serial Data (IIC*n*_SDA) | HIGH | I | When the I$^2$C module is idle or in a receiving mode, SDA defaults as an input. The unit receives data from other I$^2$C devices on SDA. The bus is assumed to be busy when SDA is detected low. |
| | | O | When writing as a master or slave, the I$^2$C module drives data on SDA synchronous to SCL. |

### 11.2.2 Detailed Signal Descriptions

SDA and SCL, described in Table 11-2, serve as a communication interconnect with other devices. All devices connected to these two signals must have open-drain or open-collector outputs. The logic AND function is performed on both of these signals with external pull-up resistors. Refer to the device hardware specifications for the electrical characteristics of these signals.

**Table 11-2. I$^2$C Interface Signal—Detailed Signal Descriptions**

| Signal | I/O | Description |
|---|---|---|
| IIC*n*_SCL | I/O | Serial clock. Performs as an input when the device is programmed as an I$^2$C slave. SCL also performs as an output when the device is programmed as an I$^2$C master. |
| | O | As outputs for the bidirectional serial clock, these signals operate as described below. |
| | | **State Meaning** Asserted/Negated—Driven along with SDA as the clock for the data. |
| | I | As inputs for the bidirectional serial clock, these signals operate as described below. |
| | | **State Meaning** Asserted/Negated—The I$^2$C unit uses this signal to synchronize incoming data on SDA. The bus is assumed to be busy when this signal is detected low. |
| IIC*n*_SDA | I/O | Serial data. Performs as an input when the device is in a receiving mode. SDA also performs as an output signal when the device is transmitting (as an I$^2$C master or a slave). |
| | O | As outputs for the bidirectional serial data, these signals operate as described below. |
| | | **State Meaning** Asserted/Negated— Data is driven. |
| | I | As inputs for the bidirectional serial data, these signals operate as described below. |
| | | **State Meaning** Asserted/Negated—Used to receive data from other devices. The bus is assumed to be busy when SDA is detected low. |

## 11.3 Memory Map/Register Definition

Table 11-3 lists the I$^2$C-specific registers and their offsets. It lists the offset, name, and a cross-reference to the complete description of each register. Note that the full register address is comprised of CCSRBAR together with the block base address and offset listed in Table 11-3. The offsets to the memory map table are defined for both I$^2$C interfaces. That is, I$^2$C1 starts at address offset 0x000, and I$^2$C2 starts at address offset 0x100. The registers for I$^2$C1 are listed in Table 11-3, but the registers for I$^2$C2 are not. Note that the registers are the same for I$^2$C2 except that the offsets change from 0x0*nn* to 0x1*nn*.

In this table and in the register figures and field descriptions, the following access definitions apply:

- Reserved fields are always ignored for the purposes of determining access type.
- R/W, R, and W (read/write, read only, and write only) indicate that all the non-reserved fields in a register have the same access type.
- w1c indicates that all of the non-reserved fields in a register are cleared by writing ones to them.
- Mixed indicates a combination of access types.
- Special is used when no other category applies. In this case the register figure and field description table should be read carefully.

All I$^2$C registers are one byte wide. Reads and writes to these registers must be byte-wide operations.

**Table 11-3. I$^2$C Memory Map**

| Offset | I$^2$C Register | Access | Reset | Section/Page |
|---|---|---|---|---|
| **I$^2$C1 Registers** | | | | |
| **Block Base Address: 0x0_3000** | | | | |
| 0x000 | I2CADR—I$^2$C address register | R/W | 0x00 | 11.3.1.1/11-6 |
| 0x004 | I2CFDR—I$^2$C frequency divider register | R/W | 0x00 | 11.3.1.2/11-6 |
| 0x008 | I2CCR—I$^2$C control register | Mixed | 0x00 | 11.3.1.3/11-7 |
| 0x00C | I2CSR—I$^2$C status register | Mixed | 0x81 | 11.3.1.4/11-9 |
| 0x010 | I2CDR—I$^2$C data register | R/W | 0x00 | 11.3.1.5/11-10 |
| 0x014 | I2CDFSRR—I$^2$C digital filter sampling rate register | R/W | 0x10 | 11.3.1.6/11-11 |
| **I$^2$C2 Registers** | | | | |
| **Block Base Address: 0x0_3000** | | | | |
| 0x100–<br>0x114 | I$^2$C2 Registers[1] | | | |

[1] I$^2$C2 has the same memory-mapped registers that are described for I$^2$C1 from 0x000 to 0x014, except the offsets range from 0x100 to 0x114.

## 11.3.1 Register Descriptions

This section describes the I$^2$C registers in detail.

### NOTE

Reserved bits should always be written with the value they returned when read. That is, the register should be programmed by reading the value, modifying appropriate fields, and writing back the value. The return value of the reserved fields should not be assumed, even though the reserved fields return zero.

This note does not apply to the I$^2$C data register (I2CDR).

### 11.3.1.1 I$^2$C Address Register (I2CADR)

Figure 11-2 shows the I2CADR register, which contains the address to which the I$^2$C interface responds when addressed as a slave. Note that this is not the address that is sent on the bus during the address-calling cycle when the I$^2$C module is in master mode.

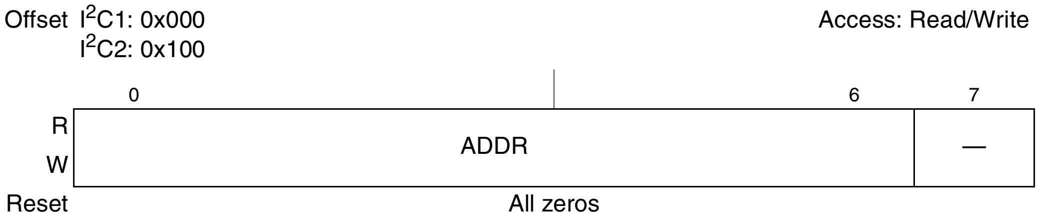


**Figure 11-2. I$^2$C Address Register (I2CADR)**

Table 11-4 describes the fields of I2CADR.

**Table 11-4. I2CADR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–6 | ADDR | Slave address. Contains the specific slave address that is used by the I$^2$C interface. Note that the default mode of the I$^2$C interface is slave mode for an address match. Note that an address match is one of the conditions that can cause I2CSR[MIF] to be set, signaling an interrupt pending condition. |
| 7 | — | Reserved |

### 11.3.1.2 I$^2$C Frequency Divider Register (I2CFDR)

Figure 11-3 shows the bits of the I$^2$C frequency divider register. Refer to application note AN2919, *Determining the I$^2$C Frequency Divider Ratio for SCL*, for additional guidance regarding the proper use of I2CFDR and I2CDFSRR.

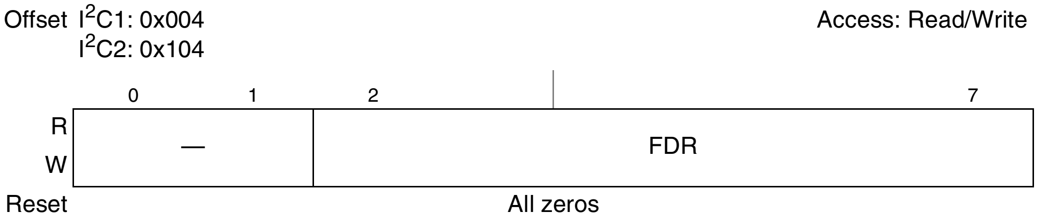


**Figure 11-3. I$^2$C Frequency Divider Register (I2CFDR)**

Table 11-5 describes the bit settings of I2CFDR. It also maps the I2CFDR[FDR] field to the clock divider values.

Table 11-5. I2CFDR Field Descriptions

<table>
<tr><th>Bits</th><th>Name</th><th>Description</th></tr>
<tr><td>0–1</td><td>—</td><td>Reserved</td></tr>
<tr><td>2–7</td><td>FDR</td><td>Frequency divider ratio. Used to prescale the clock for bit rate selection. The serial bit clock frequency of SCL is equal to one half the platform (CCB) clock divided by the designated divider. Note that the frequency divider value can be changed at any point in a program. The serial bit clock frequency divider selections are described as follows:
<table>
<tr><th>FDR</th><th>Divider (Decimal)</th><th>FDR</th><th>Divider (Decimal)</th><th>FDR</th><th>Divider (Decimal)</th></tr>
<tr><td>0x00</td><td>384</td><td>0x16</td><td>12288</td><td>0x2B</td><td>1024</td></tr>
<tr><td>0x01</td><td>416</td><td>0x17</td><td>15360</td><td>0x2C</td><td>1280</td></tr>
<tr><td>0x02</td><td>480</td><td>0x18</td><td>18432</td><td>0x2D</td><td>1536</td></tr>
<tr><td>0x03</td><td>576</td><td>0x19</td><td>20480</td><td>0x2E</td><td>1792</td></tr>
<tr><td>0x04</td><td>640</td><td>0x1A</td><td>24576</td><td>0x2F</td><td>2048</td></tr>
<tr><td>0x05</td><td>704</td><td>0x1B</td><td>30720</td><td>0x30</td><td>2560</td></tr>
<tr><td>0x06</td><td>832</td><td>0x1C</td><td>36864</td><td>0x31</td><td>3072</td></tr>
<tr><td>0x07</td><td>1024</td><td>0x1D</td><td>40960</td><td>0x32</td><td>3584</td></tr>
<tr><td>0x08</td><td>1152</td><td>0x1E</td><td>49152</td><td>0x33</td><td>4096</td></tr>
<tr><td>0x09</td><td>1280</td><td>0x1F</td><td>61440</td><td>0x34</td><td>5120</td></tr>
<tr><td>0x0A</td><td>1536</td><td>0x20</td><td>256</td><td>0x35</td><td>6144</td></tr>
<tr><td>0x0B</td><td>1920</td><td>0x21</td><td>288</td><td>0x36</td><td>7168</td></tr>
<tr><td>0x0C</td><td>2304</td><td>0x22</td><td>320</td><td>0x37</td><td>8192</td></tr>
<tr><td>0x0D</td><td>2560</td><td>0x23</td><td>352</td><td>0x38</td><td>10240</td></tr>
<tr><td>0x0E</td><td>3072</td><td>0x24</td><td>384</td><td>0x39</td><td>12288</td></tr>
<tr><td>0x0F</td><td>3840</td><td>0x25</td><td>448</td><td>0x3A</td><td>14336</td></tr>
<tr><td>0x10</td><td>4608</td><td>0x26</td><td>512</td><td>0x3B</td><td>16384</td></tr>
<tr><td>0x11</td><td>5120</td><td>0x27</td><td>576</td><td>0x3C</td><td>20480</td></tr>
<tr><td>0x12</td><td>6144</td><td>0x28</td><td>640</td><td>0x3D</td><td>24576</td></tr>
<tr><td>0x13</td><td>7680</td><td>0x29</td><td>768</td><td>0x3E</td><td>28672</td></tr>
<tr><td>0x14</td><td>9216</td><td>0x2A</td><td>896</td><td>0x3F</td><td>32768</td></tr>
<tr><td>0x15</td><td>10240</td><td></td><td></td><td></td><td></td></tr>
</table>
</td></tr>
</table>

### 11.3.1.3 I$^2$C Control Register (I2CCR)

Figure 11-4 shows the I$^2$C control register, I2CCR.

Offset I$^2$C1: 0x008
I$^2$C2: 0x108

Access: Mixed

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| R | MEN | MIEN | MSTA | MTX | TXAK | | — | BCST |
| W | | | | | | RSTA | | |
| Reset | All zeros | | | | | | | |

Figure 11-4. I$^2$C Control Register (I2CCR)

Table 11-6 describes the bit settings of the I2CCR.

Table 11-6. I2CCR Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0 | MEN | Module enable. This bit controls the software reset of the I$^2$C module.<br>0 The module is reset and disabled. When low, the interface is held in reset but the registers can still be accessed.<br>1 The I$^2$C module is enabled. This bit must be set before any other control register bits have any effect. All I$^2$C registers for slave receive or master START can be initialized before setting this bit. |
| 1 | MIEN | Module interrupt enable<br>0 Interrupts from the I$^2$C module are disabled. This does not clear any pending interrupt conditions.<br>1 Interrupts from the I$^2$C module are enabled. An interrupt occurs provided I2CSR[MIF] is also set. |
| 2 | MSTA | Master/slave mode START<br>0 When this bit is changed from one to zero, a STOP condition is generated and the mode changes from master to slave.<br>1 Cleared without generating a STOP condition when the master loses arbitration. When this bit is changed from zero to one, a START condition is generated on the bus, and master mode is selected. |
| 3 | MTX | Transmit/receive mode select. This bit selects the direction of the master and slave transfers. When configured as a slave, this bit should be set by software according to I2CSR[SRW]. In master mode, the bit should be set according to the type of transfer required. Therefore, for address cycles, this bit is always high. The MTX bit is cleared when the master loses arbitration.<br>0 Receive mode<br>1 Transmit mode |
| 4 | TXAK | Transfer acknowledge. This bit specifies the value driven onto the SDA line during acknowledge cycles for both master and slave receivers. The value of this bit only applies when the I$^2$C module is configured as a receiver, not a transmitter. It also does not apply to address cycles; when the device is addressed as a slave, an acknowledge is always sent.<br>0 An acknowledge signal (low value on SDA) is sent out to the bus at the 9th clock after receiving one byte of data.<br>1 No acknowledge signal response (high value on SDA) is sent. |
| 5 | RSTA | Repeated START. Setting this bit always generates a repeated START condition on the bus, provides the device with the current bus master. Attempting a repeated START at the wrong time (or if the bus is owned by another master), results in loss of arbitration. Note that this bit is not readable, which means if a read is performed to I2CCR[RSTA], a zero value is returned.<br>0 No START condition is generated<br>1 Generates repeated START condition |
| 6 | — | Reserved |
| 7 | BCST | Broadcast<br>0 Disables the broadcast accept capability<br>1 Enables the I$^2$C to accept broadcast messages at address zero |

### 11.3.1.4 I$^2$C Status Register (I2CSR)

The I$^2$C status register, shown in Figure 11-5, is read only with the exception of the MIF and MAL bits, which can be cleared by software. The MCF and RXAK bits are set at reset; all other I2CSR bits are cleared on reset.

Offset I$^2$C1: 0x00C
I$^2$C2: 0x10C

Access: Mixed

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| R | MCF | MAAS | MBB | MAL | BCSTM | SRW | MIF | RXAK |
| W | | | | | | | | |
| Reset | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

**Figure 11-5. I$^2$C Status Register (I2CSR)**

Table 11-7 describes the bit settings of the I2CSR.

**Table 11-7. I2CSR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | MCF | Data transfer. When one byte of data is transferred, the bit is cleared. It is set by the falling edge of the 9th clock of a byte transfer.<br>0 Byte transfer in progress. MCF is cleared under the following conditions:<br>•When I2CDR is read in receive mode or<br>•When I2CDR is written in transmit mode<br>1 Byte transfer is completed |
| 1 | MAAS | Addressed as a slave. When the value in I2CDR matches with the calling address, this bit is set. The processor is interrupted, if I2CCR[MIEN] is set. Next, the processor must check the SRW bit and set I2CCR[MTX] accordingly. Writing to the I2CCR automatically clears this bit.<br>0 Not addressed as a slave<br>1 Addressed as a slave |
| 2 | MBB | Bus busy. Indicates the status of the bus. When a START condition is detected, MBB is set. If a STOP condition is detected, it is cleared.<br>0 I$^2$C bus is idle<br>1 I$^2$C bus is busy |
| 3 | MAL | Arbitration lost. Automatically set when the arbitration procedure is lost. Note that the device does not automatically retry a failed transfer attempt.<br>0 Arbitration is not lost. Can only be cleared by software<br>1 Arbitration is lost |
| 4 | BCSTM | Broadcast match<br>0 There has not been a broadcast match.<br>1 The calling address matches with the broadcast address instead of the programmed slave address. This also sets if this I$^2$C drives an address of all 0s and broadcast mode is enabled. |
| 5 | SRW | Slave read/write. When MAAS is set, SRW indicates the value of the R/W command bit of the calling address, which is sent from the master.<br>0 Slave receive, master writing to slave<br>1 Slave transmit, master reading from slave. This bit is valid only when both of the following conditions are true:<br>•A complete transfer occurred and no other transfers have been initiated.<br>•The I$^2$C interface is configured as a slave and has an address match.<br>By checking this bit, the processor can select slave transmit/receive mode according to the command of the master. |

Table 11-7. I2CSR Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 6 | MIF | Module interrupt. The MIF bit is set when an interrupt is pending, causing a processor interrupt request (provided I2CCR[MIEN] is set). The interrupts for I$^2$C1 and I$^2$C2 are combined into one interrupt, which is sourced by the dual I$^2$C controller.<br>0 No interrupt is pending. Can be cleared only by software.<br>1 Interrupt is pending. MIF is set when one of the following events occurs:<br>•One byte of data is transferred (set at the falling edge of the 9th clock).<br>•The value in I2CADR matches with the calling address in slave-receive mode.<br>•Arbitration is lost. |
| 7 | RXAK | Received acknowledge. The value of SDA during the reception of acknowledge bit of a bus cycle. If the received acknowledge bit (RXAK) is low, it indicates that an acknowledge signal has been received after the completion of eight bits of data transmission on the bus. If RXAK is high, it means no acknowledge signal has been detected at the 9th clock.<br>0 Acknowledge received<br>1 No acknowledge received |

### 11.3.1.5 I$^2$C Data Register (I2CDR)

The I2C data register is shown in Figure 11-6.

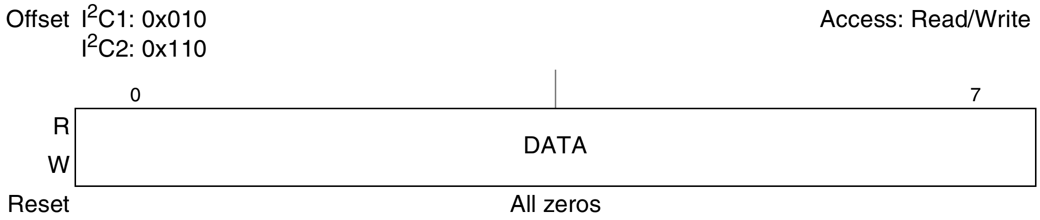


Figure 11-6. I$^2$C Data Register (I2CDR)

Table 11-8 shows the bit descriptions for I2CDR.

Table 11-8. I2CDR Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–7 | DATA | Transmission starts when an address and the R/W bit are written to the data register and the I$^2$C interface performs as the master. A data transfer is initiated when data is written to the I2CDR. The most significant bit is sent first in both cases. In master receive mode, reading the data register allows the read to occur, but also allows the I$^2$C module to receive the next byte of data on the I2C interface. In slave mode, the same function is available after it is addressed. Note that in both master receive and slave receive modes, the very first read is always a dummy read. |

### 11.3.1.6 Digital Filter Sampling Rate Register (I2CDFSRR)

The digital filter sampling rate register (I2CDFSRR) is shown in Figure 11-7. Refer to application note AN2919, *Determining the I$^2$C Frequency Divider Ratio for SCL*, for additional guidance regarding the proper use of I2CFDR and I2CDFSRR.

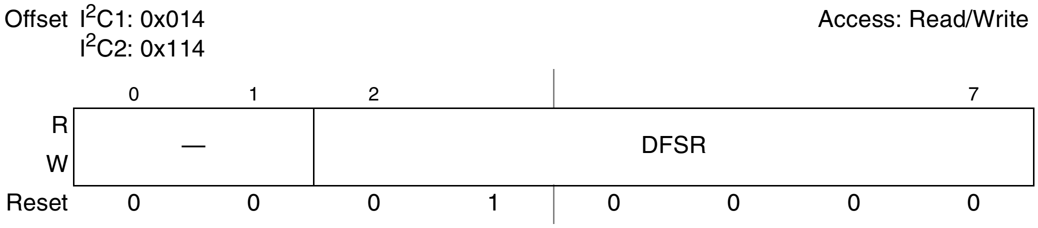


**Figure 11-7. I$^2$C Digital Filter Sampling Rate Register (I2CDFSRR)**

Table 11-9 shows the field descriptions for I2CDFSRR.

**Table 11-9. I2CDFSRR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–1 | — | Reserved |
| 2–7 | DFSR | Digital filter sampling rate. To assist in filtering out signal noise, the sample rate is programmed. This field is used to prescale the frequency at which the digital filter takes samples from the I$^2$C bus. The resulting sampling rate is calculated by dividing one half the platform (CCB clock) frequency by the non-zero value of DFSR. |

## 11.4 Functional Description

The I$^2$C unit always performs as a slave receiver as a default, unless explicitly programmed to be a master or slave transmitter. After the boot sequencer has completed (when powered up in boot sequencer mode), the I$^2$C interface performs as a slave receiver.

Note that the boot sequencer only functions from the I$^2$C1 interface; the I$^2$C2 interface cannot be used for this purpose.

### 11.4.1 Transaction Protocol

A standard I$^2$C transfer consists of the following:

- START condition
- Slave target address transmission
- Data transfer
- STOP condition

Figure 11-8 shows the interaction of these four parts with the calling address, data byte, and new calling address components of the I$^2$C protocol. The details of the protocol are described in the following sections.

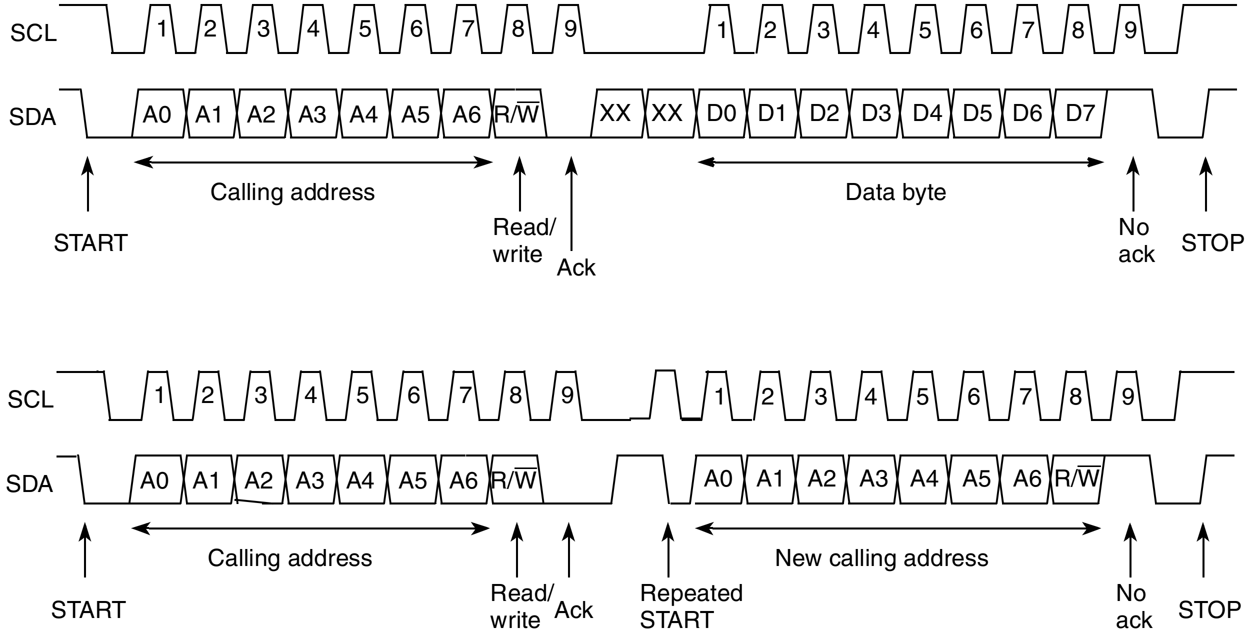


**Figure 11-8. I$^2$C Interface Transaction Protocol**

### 11.4.1.1 START Condition

When the I$^2$C bus is not engaged (both SDA and SCL lines are at logic high), a master can initiate a transfer by sending a START condition. As shown in Figure 11-8, a START condition is defined as a high-to-low transition of SDA while SCL is high. This condition denotes the beginning of a new data transfer. Each data transfer can contain several bytes and awakens all slaves. The START condition is initiated by a software write that sets I2CCR[MSTA].

### 11.4.1.2 Slave Address Transmission

The first byte of data is transferred by the master immediately after the START condition is the slave address. This is a seven-bit calling address followed by a R/$\overline{W}$ bit, which indicates the direction of the data being transferred to the slave. Each slave in the system has a unique address. In addition, when the I$^2$C module is operating as a master, it must not transmit an address that is the same as its slave address. An I$^2$C device cannot be master and slave at the same time; if this is attempted, the results are boundedly undefined.

Only the slave with a calling address that matches the one transmitted by the master responds by returning an acknowledge bit (pulling the SDA signal low at the 9th clock) as shown in Figure 11-8. If no slave acknowledges the address, the master should generate a STOP condition or a repeated START condition.

When slave addressing is successful (and SCL returns to zero), the data transfer can proceed on a byte-to-byte basis in the direction specified by the R/$\overline{W}$ bit sent by the calling master.

The I$^2$C module responds to a general call (broadcast) command when I2CCR[BCST] is set. A broadcast address is always zero; however the I$^2$C module does not check the R/$\overline{\text{W}}$ bit. The second byte of the broadcast message is the master address. Because the second byte is automatically acknowledged by hardware, the receiver device software must verify that the broadcast message is intended for itself by reading the second byte of the message. If the master address is for another receiver device and the third byte is a write command, software can ignore the third byte during the broadcast. If the master address is for another receiver device and the third byte is a read command, software must write 0xFF to I2CDR with I2CCR[TXAK] = 1, so that it does not interfere with the data written from the addressed device.

Each data byte is 8 bits long. Data bits can be changed only while SCL is low and must be held stable while SCL is high, as shown in Figure 11-8. There is one clock pulse on SCL for each data bit, and the most significant bit (msb) is transmitted first. Each byte of data must be followed by an acknowledge bit, which is signaled from the receiving device by pulling the SDA line low at the 9th clock. Therefore, one complete data byte transfer takes 9 clock pulses. Several bytes can be transferred during a data transfer session.

If the slave receiver does not acknowledge the master, the SDA line must be left high by the slave. The master can then generate a stop condition to abort the data transfer or a START condition (repeated START) to begin a new calling.

If the master receiver does not acknowledge the slave transmitter after a byte of transmission, the slave interprets that the end-of-data has been reached. Then the slave releases the SDA line for the master to generate a STOP or a START condition.

#### 11.4.1.3 Repeated START Condition

Figure 11-8 shows a repeated START condition, which is generated without a STOP condition that can terminate the previous transfer. The master uses this method to communicate with another slave or with the same slave in a different mode (transmit/receive mode) without releasing the bus.

#### 11.4.1.4 STOP Condition

The master can terminate the transfer by generating a STOP condition to free the bus. A STOP condition is defined as a low-to-high transition of the SDA signal while SCL is high. For more information, see Figure 11-8. Note that a master can generate a STOP even if the slave has transmitted an acknowledge bit, at which point the slave must release the bus. The STOP condition is initiated by a software write that clears I2CCR[MSTA].

As described in Section 11.4.1.3, “Repeated START Condition,” the master can generate a START condition followed by a calling address without generating a STOP condition for the previous transfer. This is called a repeated START condition.

#### 11.4.1.5 Protocol Implementation Details

The following sections give details of how aspects of the protocol are implemented in this I$^2$C module.

#### 11.4.1.5.1 Transaction Monitoring—Implementation Details

The different conditions of the I$^2$C data transfers are monitored as follows:

- START conditions are detected when an SDA fall occurs while SCL is high.
- STOP conditions are detected when an SDA rise occurs while SCL is high.
- Data transfers in progress are canceled when a STOP condition is detected or if there is a slave address mismatch. Cancellation of data transactions resets the clock module.
- The bus is detected to be busy upon the detection of a START condition, and idle upon the detection of a STOP condition.

#### 11.4.1.5.2 Control Transfer—Implementation Details

The I$^2$C module contains logic that controls the output to the serial data (SDA) and serial clock (SCL) lines of the I$^2$C. The SCL output is pulled low as determined by the internal clock generated in the clock module. The SDA output can only change at the midpoint of a low cycle of the SCL, unless it is performing a START, STOP, or restart condition. Otherwise, the SDA output is held constant.

The SDA signal is pulled low when one or more of the following conditions are true in either master or slave mode:

- Master mode
  - Data bit (transmit)
  - Ack bit (receive)
  - START condition
  - STOP condition
  - Restart condition
- Slave mode
  - Acknowledging address match
  - Data bit (transmit)
  - Ack bit (receive)

The SCL signal corresponds to the internal SCL signal when one or more of the following conditions are true in either master or slave mode:

- Master mode
  - Bus owner
  - Lost arbitration
  - START condition
  - STOP condition
  - Restart condition begin
  - Restart condition end
- Slave mode
  - Address cycle
  - Transmit cycle

  - Ack cycle

### 11.4.1.6 Address Compare—Implementation Details

Address compare block determines if a slave has been properly addressed, either by its slave address or by the general broadcast address (which addresses all slaves). The three performed address comparisons are described as follows:

- Whether a broadcast message has been received, to update the I2CSR
- Whether the module has been addressed as a slave, to update the I2CSR and to generate an interrupt
- If the address transmitted by the current master matches the general broadcast address

## 11.4.2 Arbitration Procedure

The I$^2$C interface is a true multiple-master bus that allows more than one master device to be connected on it. If two or more masters simultaneously try to control the bus, each master’s clock synchronization procedure (including the I$^2$C module) determines the bus clock—the low period is equal to the longest clock low period and the high is equal to the shortest one among the masters. A bus master loses arbitration if it transmits a logic 1 on SDA while another master transmits a logic 0. The losing masters immediately switch to slave-receive mode and stop driving the SDA line. In this case, the transition from master to slave mode does not generate a STOP condition. Meanwhile, the I$^2$C unit sets the I2CSR[MAL] status bit to indicate the loss of arbitration and, as a slave, services the transaction if it is directed to itself.

If the I$^2$C module is enabled in the middle of an ongoing byte transfer, the interface behaves as follows:

- Slave mode—The I$^2$C module ignores the current transfer on the bus and starts operating whenever a subsequent START condition is detected.
- Master mode—The I$^2$C module cannot tell whether the bus is busy; therefore, if a START condition is initiated, the current bus cycle can be corrupted. This ultimately results in the current bus master of the I$^2$C interface losing arbitration, after which bus operations return to normal.

### 11.4.2.1 Arbitration Control

The arbitration control block controls the arbitration procedure of the master mode. A loss of arbitration occurs whenever the master detects a 0 on the external SDA line while attempting to drive a 1, tries to generate a START or restart at an inappropriate time, or detects an unexpected STOP request on the line.

In master mode, arbitration by the master is lost (and I2CSR[MAL] is set) under the following conditions:

- SDA samples low when the master drives high during an address or data-transmit cycle (transmit).
- SDA samples low when the master drives high during a data-receive cycle of the acknowledge (Ack) bit (receive).
- A START condition is attempted when the bus is busy.
- A repeated START condition is requested in slave mode.
- A start condition is attempted when the requesting device is not the bus owner
- Unexpected STOP condition detected

Note that the I$^2$C module does not automatically retry a failed transfer attempt.

## 11.4.3 Handshaking

The clock synchronization mechanism can be used as a handshake in data transfer. Slave devices can hold SCL low after completion of a 1-byte transfer (9 bits). In such cases, it halts the bus clock and forces the master clock into wait states until the slave releases the SCL line.

## 11.4.4 Clock Control

The clock control block handles requests from the clock signal for transferring and controlling data for multiple tasks.

A 9-cycle data transfer clock is requested for the following conditions:

- Master mode
  - Transmit slave address after START condition
  - Transmit slave address after restart condition
  - Transmit data
  - Receive data
- Slave mode
  - Transmit data
  - Receive data
  - Receive slave address after START or restart condition

### 11.4.4.1 Clock Synchronization

Due to the wire AND logic on the SCL line, a high-to-low transition on the SCL line affects all devices connected on the bus. The devices begin counting their low period when the master drives the SCL line low. After a device has driven SCL low, it holds the SCL line low until the clock high state is reached. However, the change of low-to-high in a device clock may not change the state of the SCL line if another device is still within its low period. Therefore, the synchronized clock signal, SCL, is held low by the device with the longest low period. Devices with shorter low periods enter a high wait state during this time. When all devices concerned have counted off their low period, the synchronized SCL line is released and pulled high. Then there is no difference between the devices' clocks and the state of the SCL line, and all the devices begin counting their high periods. The first device to complete its high period pulls the SCL line low again.

### 11.4.4.2 Input Synchronization and Digital Filter

The following sections describes the synchronizing of the input signals, and the filtering of the SCL and SDA lines in detail.

#### 11.4.4.2.1 Input Signal Synchronization

The input synchronization block synchronizes the input SCL and SDA signals to the system clock and detects transitions of these signals.

#### 11.4.4.2.2 Filtering of SCL and SDA Lines

The SCL and SDA inputs are filtered to eliminate noise. Three consecutive samples of the SCL and SDA lines are compared to a pre-determined sampling rate. If they are all high, the output of the filter is high. If they are all low, the output is low. If they are any combination of highs and lows, the output is whatever the value of the line was in the previous clock cycle.

The sampling rate is equal to a binary value stored in the frequency register I2CDFSRR. The duration of the sampling cycle is controlled by a down counter. This allows a software write to the frequency register to control the filtered sampling rate.

### 11.4.4.3 Clock Stretching

Slaves can use the clock synchronization mechanism to slow down the transfer bit rate. After the master has driven the SCL line low, the slave can drive SCL low for the required period and then release it. If the slave SCL low period is greater than the master SCL low period, then the resulting SCL bus signal low period is stretched.

## 11.4.5 Boot Sequencer Mode

If boot sequencer mode is selected on POR (by the settings on the cfg_boot_seq[0:1] reset configuration signals, as described in Section 4.4.3.11, “Boot Sequencer Configuration”), the I$^2$C1 module communicates with one or more EEPROMs through the I$^2$C interface on IIC1_SCL and IIC1_SDA. The boot sequencer accesses the I$^2$C1 serial ROM device at a serial bit clock frequency equal to the platform (CCB) clock frequency divided by 2560. The EEPROM(s) can be programmed to initialize one or more configuration registers of this integrated device.

If the boot sequencer is enabled for normal I$^2$C addressing mode, the I$^2$C interface initiates the following sequence during reset:

1. Generate RESET sequence (START then 9 SCL cycles) to the EEPROM twice. This clears any transactions that may have been in progress prior to the reset.
2. Generate START
3. Transmit 0xA0 which is the 7-bit calling address (0b101_0000) with a write command appended (0 as the least significant bit).
4. Transmit 0x00 which is the 8-bit starting address
5. Generate a repeated START
6. Transmit 0xA1 which is the 7-bit calling address (0b101_0000) with a read command appended (1 as the least significant bit).
7. Receive 256 bytes of data from the EEPROM (unless the CONT bit is cleared in the data structure).
8. Generate a repeated START
9. Transmit 0xA2 which is the 7-bit calling address of the second target (0b101_0001) with a write command appended (0 as the least significant bit).
10. Transmit 0x00 which is the 8-bit starting address for the second target.
11. Generate a repeated START

12. Transmit 0xA3 which is the 7-bit calling address (0b101_0001) with a read command appended (1 as the least significant bit).
13. Receive another 256 bytes of data from the second EEPROM (unless the CONT bit is cleared in the data structure).

The sequence repeats with successive targets until the CONT bit in the data structure is cleared and the CRC check is executed. If the last register is not detected (that is, the CONT bit is never cleared) before wrapping back to the first address, an error condition is detected, causing the device to hang and the $\overline{\text{HRESET\_REQ}}$ signal to assert externally. The I$^2$C module continues to read from the EEPROM(s) as long as the continue (CONT) bit is set in the EEPROM(s). The CONT bit resides in the address/attributes field that is transferred from the EEPROM, as described in Section 11.4.5.1, "EEPROM Calling Address." There should be no other I$^2$C traffic when the boot sequencer is active.

The boot sequencer mode also supports an extension of the standard I$^2$C interface that uses more address bits to allow for EEPROM devices that have more than 256 bytes, and this extended addressing mode is selectable during POR with a different encoding on the cfg_boot_seq[0:1] reset configuration signals. In this mode, only one EEPROM device may be used, and the maximum number of registers is limited by the size of the EEPROM. If the boot sequencer is enabled for extended I$^2$C addressing mode, the I$^2$C interface initiates the following sequence during reset:

1. Generate RESET sequence (START then 9 SCL cycles) to the EEPROM twice. This clears any transactions that may have been in progress prior to the reset.
2. Generate START
3. Transmit 0xA0 which is the 7-bit calling address (0b101_0000) with a write command appended (0 as the least significant bit).
4. Transmit 0x00 which is the high-order starting address
5. Transmit 0x00 which is the low-order starting address
6. Generate a repeated START
7. Transmit 0xA1 which is the 7-bit calling address (0b101_0000) with a read command appended (1 as the least significant bit).
8. Receive data continuously from the EEPROM until the CONT bit is cleared and the CRC check is executed. See Section 11.4.5.2, "EEPROM Data Format," for more information.

Note that as described in Section 4.4.3.11, "Boot Sequencer Configuration," the default value for the cfg_boot_seq[0:1] reset configuration pins is 0b11, which corresponds to the I$^2$C boot sequencer being disabled at power-up.

### 11.4.5.1 EEPROM Calling Address

The MPC8536E uses 0b101_0000 for the EEPROM calling address. The first EEPROM to be addressed must be programmed to respond to this address, or an error is generated. If more EEPROMs are used, they are addressed in sequential order.

### 11.4.5.2 EEPROM Data Format

The I$^2$C module expects that a particular data format be used for data in the EEPROM. A preamble should be the first 3 bytes programmed into the EEPROM. It should have a value of 0xAA55AA. The I$^2$C module checks to ensure that this preamble is correctly detected before proceeding further. Following the preamble, there should be a series of configuration registers (known as register preloads) programmed into the EEPROM. Each configuration register should be programmed according to a particular format, as shown in Figure 11-9. The first 3 bytes hold the attributes and address offset, as follows. The attributes contained are alternate configuration space (ACS), byte enables, and continue (CONT). The boot sequencer expects the address offset to be a 32-bit (word) offset, that is, the 2 low-order bits are not included in the boot sequencer command. For example, to access LAWBAR0 (byte offset of 0x00C08), the boot sequencer ADDR[0:17] should be set to 0x00302.

After the first 3 bytes, 4 bytes of data should hold the desired value of the configuration register, regardless of the size of the transaction. Byte enables should be asserted for any byte that is written to the configuration register, and they should be asserted contiguously, creating a 1-, 2-, or 4-byte write to a register. The boot sequencer assumes that a big-endian address is stored in the EEPROM. In addition, byte enable bit 0 (bit 1 of the byte) corresponds to the most-significant byte of data (data[0:7]), and byte enable bit 3 (bit 4 of the byte) corresponds to the LSB of data (data[24:31]).

By setting ACS, an alternate configuration space address is prepended to the write request from the boot sequencer. Otherwise, CCSRBAR is prepended to the EEPROM address.

If CONT is cleared, the first 3 bytes, including ACS, the byte enables, and the address, must also be cleared. Also, the data contains the final cyclic redundancy check (CRC). A CRC-32 algorithm is used to check the integrity of the data. The polynomial used is:

$$1 + x^1 + x^2 + x^4 + x^5 + x^7 + x^8 + x^{10} + x^{11} + x^{12} + x^{16} + x^{22} + x^{23} + x^{26} + x^{32}$$

CRC values are calculated using the above polynomial with a start value of 0xFFFF_FFFF and an XOR with 0x0000_0000. The CRC should cover all bytes stored in the EEPROM prior to the CRC. This includes the preamble, all register preloads, and the first 3 bytes of the last 7-byte preload (which should be all zeros). If a preamble or CRC fail is detected, the device hangs and the external $\overline{\text{HRESET\_REQ}}$ signal asserts. If there is a preamble fail, the boot sequencer may continue to pull I$^2$C pins low until a hard reset occurs.

| 0 | 1 4 | 5 | 6 7 |
|---|---|---|---|
| ACS | BYTE_EN | CONT | ADDR[0–1] |
| ADDR[2–9] | | | |
| ADDR[10–17] | | | |
| DATA[0–7] | | | |
| DATA[8–15] | | | |
| DATA[16–23] | | | |
| DATA[24–31] | | | |

**Figure 11-9. EEPROM Data Format for One Register Preload Command**

Figure 11-10 shows an example of the EEPROM contents, including the preamble, data format, and CRC.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | Preamble |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | |
| ACS | BYTE_EN | | | | 1 | ADDR[0–1] | | First Configuration Preload Command |
| ADDR[2–9] | | | | | | | | |
| ADDR[10–17] | | | | | | | | |
| DATA[0–7] | | | | | | | | |
| DATA[8–15] | | | | | | | | |
| DATA[16–23] | | | | | | | | |
| DATA[24–31] | | | | | | | | |
| ACS | BYTE_EN | | | | 1 | ADDR[0–1] | | Second Configuration Preload Command |
| ADDR[2–9] | | | | | | | | |
| ADDR[10–17] | | | | | | | | |
| DATA[0–7] | | | | | | | | |
| DATA[8–15] | | | | | | | | |
| DATA[16–23] | | | | | | | | |
| DATA[24–31] | | | | | | | | |
| . . . | | | | | | | | |
| ACS | BYTE_EN | | | | 1 | ADDR[0–1] | | Last Configuration Preload Command |
| ADDR[2–9] | | | | | | | | |
| ADDR[10–17] | | | | | | | | |
| DATA[0–7] | | | | | | | | |
| DATA[8–15] | | | | | | | | |
| DATA[16–23] | | | | | | | | |
| DATA[24–31] | | | | | | | | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | End Command |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

**Figure 11-10. EEPROM Contents**

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |
|---|---|---|---|---|---|---|---|---|
| CRC[0–7] | | | | | | | | Cyclic Redundancy Check |
| CRC[8–15] | | | | | | | | |
| CRC[16–23] | | | | | | | | |
| CRC[24–31] | | | | | | | | |

**Figure 11-10. EEPROM Contents (continued)**

## 11.5 Initialization/Application Information

This section describes some programming guidelines recommended for the I$^2$C interface. Figure 11-11 is a recommended flowchart for I$^2$C interrupt service routines.

The I$^2$C registers in this chapter are shown in big-endian format. If the system is in little-endian mode, software must swap the bytes appropriately. This appropriate byte swapping is needed as I$^2$C registers are byte registers. Also, an **msync** assembly instruction must be executed after each I$^2$C register read/write access to guarantee in-order execution.

The I$^2$C controller does not guarantee its recovery from all illegal I$^2$C bus activity. In addition, a malfunctioning device may hold the bus captive. A good programming practice is for software to rely on a watchdog timer to help recover from I$^2$C bus hangs. The recovery routine should also handle the case when the status bits returned after an interrupt are not consistent with what was expected due to illegal I$^2$C bus protocol behavior.

### 11.5.1 Initialization Sequence

A hard reset initializes all the I$^2$C registers to their default states. The following initialization sequence initializes the I$^2$C unit:

1. All I$^2$C registers must be located in a cache-inhibited page.
2. Update I2CFDR[FDR] and select the required division ratio to obtain the SCL frequency from the CCB (platform) clock. Note that the platform frequency must first be divided by two; see Section 11.3.1.2, “I$^2$C Frequency Divider Register (I2CFDR),” for more details.
3. Update I2CADR to define the slave address for this device.
4. Modify I2CCR to select master/slave mode, transmit/receive mode, and interrupt-enable or disable.
5. Set the I2CCR[MEN] to enable the I$^2$C interface.

### 11.5.2 Generation of START

After initialization, the following sequence can be used to generate START:

1. If the device is connected to a multimaster I$^2$C system, test the state of I2CSR[MBB] to check whether the serial bus is free (I2CSR[MBB] = 0) before switching to master mode.
2. Select master mode (set I2CCR[MSTA]) to transmit serial data and select transmit mode (set I2CCR[MTX]) for the address cycle.

3. Write the slave address being called into I2CDR. The data written to I2CDR[0–6] comprises the slave calling address. I2CCR[MTX] indicates the direction of transfer (transmit/receive) required from the slave.

The scenario above assumes that the I$^2$C interrupt bit (I2CSR[MIF]) is cleared. If MIF is set at any time, an I$^2$C interrupt is generated (provided interrupt reporting is enabled with I2CCR[MIEN] =1) so that the I$^2$C interrupt handler can handle the interrupt. Note that the interrupts for I$^2$C1 and I$^2$C2 are combined into one interrupt, which is sourced by the dual I$^2$C controller.

### 11.5.3 Post-Transfer Software Response

Transmission or reception of a byte automatically sets the data transferring bit (I2CSR[MCF]), which indicates that one byte has been transferred. The I$^2$C interrupt bit (I2CSR[MIF]) is also set and an interrupt is generated to the processor if the interrupt function is enabled during the initialization sequence (I2CCR[MIEN] is set). In the interrupt handler, software must take the following steps:

1. Clear I2CSR[MIF]
2. Read the contents of the I$^2$C data register (I2CDR) in receive mode or write to I2CDR in transmit mode. Note that this causes I2CSR[MCF] to be cleared. See Section 11.5.8, “Interrupt Service Routine Flowchart.”

When an interrupt occurs at the end of the address cycle, the master remains in transmit mode. If master receive mode is required, I2CCR[MTX] must be toggled at this stage. See Section 11.5.8, “Interrupt Service Routine Flowchart.”

If the interrupt function is disabled, software can service the I2CDR in the main program by monitoring I2CSR[MIF]. In this case, I2CSR[MIF] must be polled rather than I2CSR[MCF] because MCF behaves differently when arbitration is lost. Note that interrupt or other bus conditions may be detected before the I$^2$C signals have time to settle. Thus, when polling I2CSR[MIF] (or any other I2CSR bits), software delays may be needed in order to give the I$^2$C signals sufficient time to settle.

During slave-mode address cycles (I2CSR[MAAS] is set), I2CSR[SRW] should be read to determine the direction of the subsequent transfer and I2CCR[MTX] should be programmed accordingly. For slave-mode data cycles (MAAS is cleared), I2CSR[SRW] is not valid and I2CCR[MTX] must be read to determine the direction of the current transfer. See Section 11.5.8, “Interrupt Service Routine Flowchart,” for more details.

### 11.5.4 Generation of STOP

A data transfer ends with a STOP condition generated by the master device. A master transmitter can generate a STOP condition after all the data has been transmitted.

If a master receiver wants to terminate a data transfer, it must inform the slave transmitter by not acknowledging the last byte of data (by setting the transmit acknowledge bit (I2CCR[TXAK])) before reading the next-to-last byte of data. At this time, the next-to-last byte of data has already been transferred on the I$^2$C interface, so the last byte does not receive the data acknowledge (because I2CCR[TXAK] is set). Before the interrupt service routine reads the last byte of data, a STOP condition must first be generated.

The I$^2$C controller automatically generates a STOP if I2CCR[TXAK] is set. Therefore, I2CCR[TXAK] must be set before allowing the I$^2$C module to receive the last data byte on the I$^2$C bus. Eventually, I2CCR[TXAK] needs to be cleared again for subsequent I$^2$C transactions. This can be accomplished when setting up the I2CCR for the next transfer.

### 11.5.5 Generation of Repeated START

At the end of a data transfer, if the master still wants to communicate on the bus, it can generate another START condition followed by another slave address without first generating a STOP condition. This is accomplished by setting I2CCR[RSTA].

### 11.5.6 Generation of SCL When SDA Low

It is sometimes necessary to force the I$^2$C module to become the I$^2$C bus master out of reset and drive SCL (even though SDA may already be driven, which indicates that the bus is busy). This can occur when a system reset does not cause all I$^2$C devices to be reset. Thus, SDA can be driven low by another I$^2$C device while this I$^2$C module is coming out of reset and stays low indefinitely. The following procedure can be used to force this I$^2$C module to generate SCL so that the device driving SDA can finish its transaction:

1. Disable the I$^2$C module and set the master bit by setting I2CCR to 0x20
2. Enable the I$^2$C module by setting I2CCR to 0xA0
3. Read the I2CDR
4. Return the I$^2$C module to slave mode by setting I2CCR to 0x80

### 11.5.7 Slave Mode Interrupt Service Routine

In the slave interrupt service routine, the module addressed as a slave should be tested to check if a calling of its own address has been received. If I2CSR[MAAS] is set, software should set the transmit/receive mode select bit (I2CCR[MTX]) according to the R/$\overline{\text{W}}$ command bit (I2CSR[SRW]). Writing to I2CCR clears MAAS automatically. MAAS is read as set only in the interrupt handler at the end of that address cycle where an address match occurred; interrupts resulting from subsequent data transfers clear MAAS. A data transfer can then be initiated by writing to I2CDR for slave transmits or dummy reading from I2CDR in slave-receive mode. The slave drives SCL low between byte transfers. SCL is released when the I2CDR is accessed in the required mode.

#### 11.5.7.1 Slave Transmitter and Received Acknowledge

In the slave transmitter routine, the received acknowledge bit (I2CSR[RXAK]) must be tested before sending the next byte of data. The master signals an end-of-data by not acknowledging the data transfer from the slave. When no acknowledge is received (I2CSR[RXAK] is set), the slave transmitter interrupt routine must clear I2CCR[MTX] to switch the slave from transmitter to receiver mode. A dummy read of I2CDR then releases SCL so that the master can generate a STOP condition. See Section 11.5.8, “Interrupt Service Routine Flowchart.”

### 11.5.7.2 Loss of Arbitration and Forcing of Slave Mode

When a master loses arbitration the following conditions all occur:

- I2CSR[MAL] is set
- I2CCR[MSTA] is cleared (changing the master to slave mode)
- An interrupt occurs (if enabled) at the falling edge of the 9th clock of this transfer

Thus, the slave interrupt service routine should first test I2CSR[MAL] and software should clear it if it is set. See Section 11.4.2.1, “Arbitration Control,” for more information.

## 11.5.8 Interrupt Service Routine Flowchart

Figure 11-11 shows an example algorithm for an I$^2$C interrupt service routine. Deviation from the flowchart may result in unpredictable I$^2$C bus behavior. However, in the slave receive mode the interrupt service routine may need to set I2CCR[TXAK] when the next-to-last byte is to be accepted. It is recommended that an **msync** instruction follow each I$^2$C register read or write to guarantee in-order instruction execution.

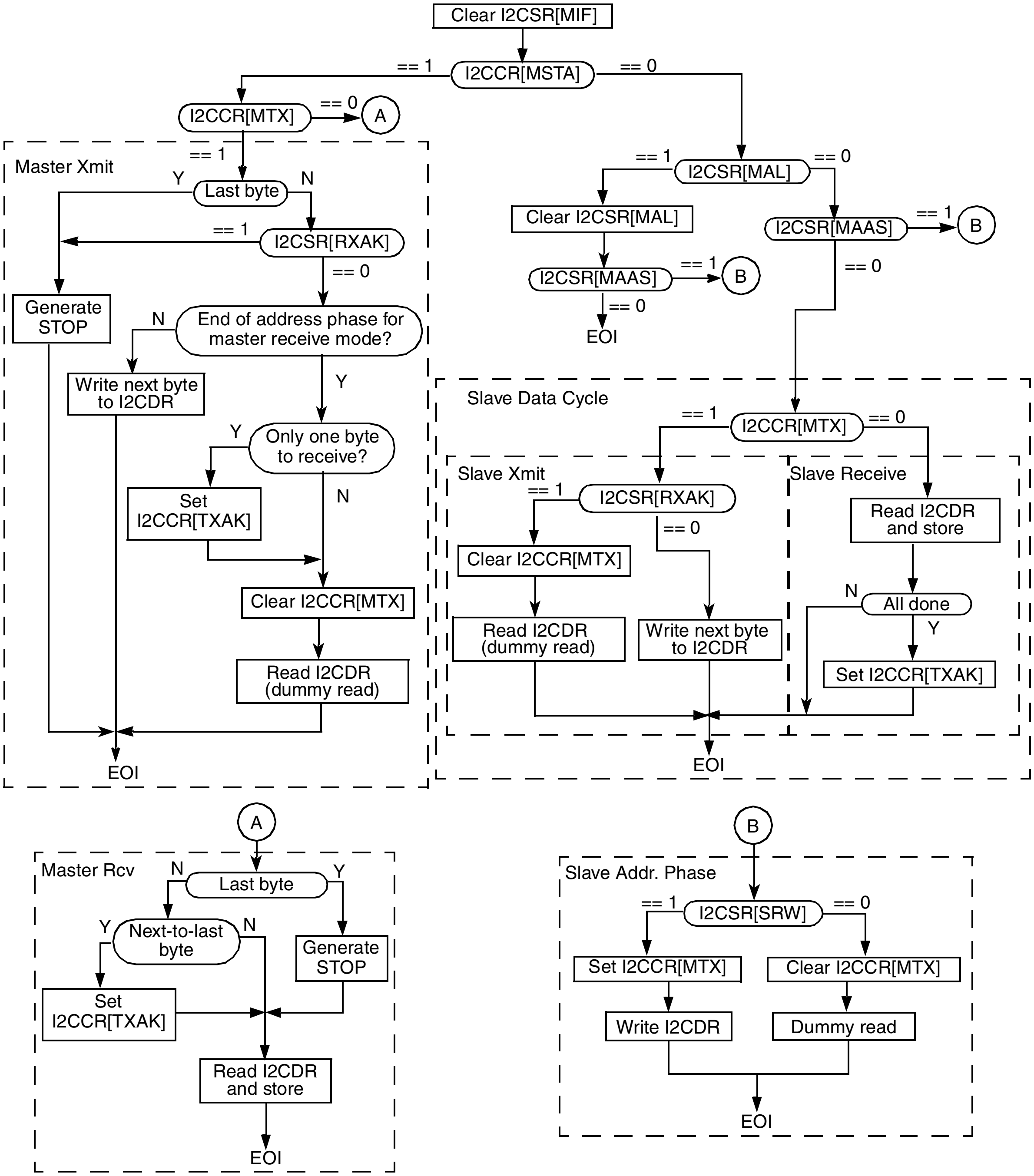


**Figure 11-11. Example I²C Interrupt Service Routine Flowchart**

# Chapter 12
# DUART

This chapter describes the dual universal asynchronous receiver/transmitters (DUART). It describes the functional operation, the initialization sequence, and the programming details for the DUART registers and features.

## 12.1 Overview

The DUART consists of two universal asynchronous receiver/transmitters (UARTs). The UARTs act independently; all references to UART refer to one of these receiver/transmitters. Each UART is clocked by the platform (CCB) clock. The DUART programming model is compatible with the PC16552D.

The UART interface is point to point, meaning that only two UART devices are attached to the connecting signals. As shown in Figure 12-1, each UART module consists of the following:

- Receive and transmit buffers
- Clear to send ($\overline{\text{CTS}}$) input port and request to send ($\overline{\text{RTS}}$) output port for data flow control
- 16-bit counter for baud rate generation
- Interrupt control logic

### 12.1.1 Features

The DUART includes these distinctive features:

- Full-duplex operation
- Programming model compatible with original PC16450 UART and PC16550D (improved version of PC16450 that also operates in FIFO mode)
- PC16450 register reset values
- FIFO mode for both transmitter and receiver, providing 16-byte FIFOs
- Serial data encapsulation and decapsulation with standard asynchronous communication bits (START, STOP, and parity)
- Maskable transmit, receive, line status, and modem status interrupts
- Software-programmable baud generators that divide the platform clock by 1 to ($2^{16}$ – 1) and generate a 16x clock for the transmitter and receiver engines

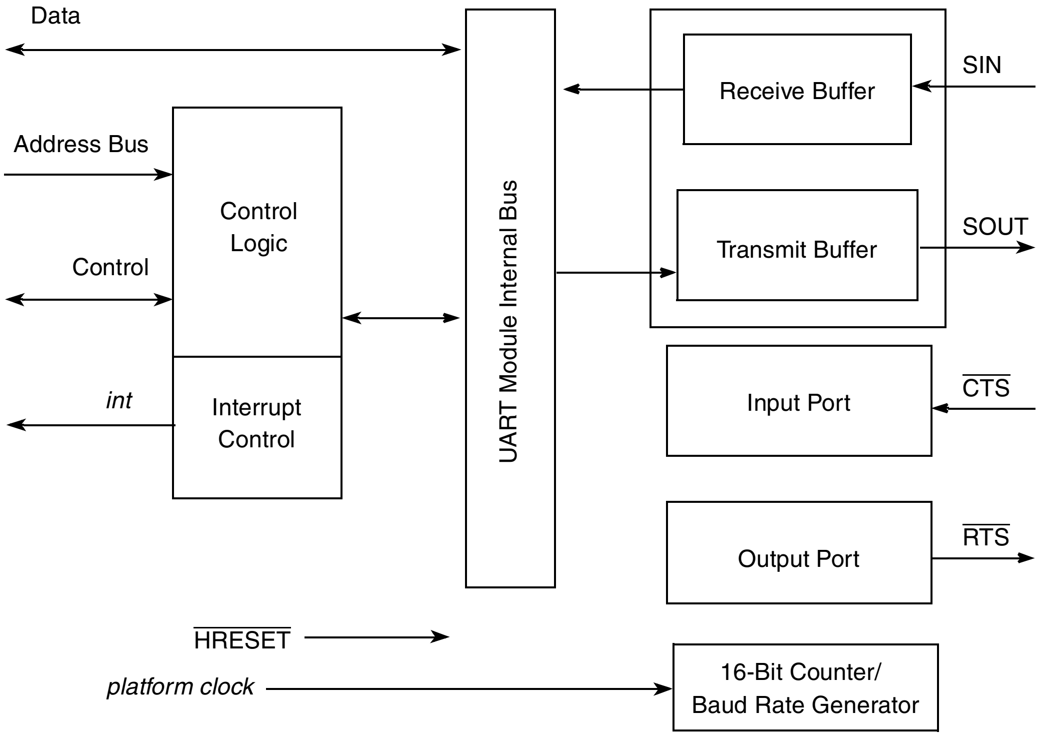


**Figure 12-1. UART Block Diagram**

- Clear to send (CTS) and ready to send (RTS) modem control functions
- Software-selectable serial interface data format (data length, parity, 1/1.5/2 STOP bit, baud rate)
- Line and modem status registers
- Line-break detection and generation
- Internal diagnostic support, local loopback, and break functions
- Prioritized interrupt reporting
- Overrun, parity, and framing error detection

## 12.1.2 Modes of Operation

The communication channel provides a full-duplex asynchronous receiver and transmitter using an operating frequency derived from the platform clock.

The transmitter accepts parallel data from a write to the transmitter holding register (UTHR). In FIFO mode, the data is placed directly into an internal transmitter shift register of the transmitter FIFO. The transmitter converts the data to a serial bit stream inserting the appropriate start, stop, and optional parity bits. Finally, it outputs a composite serial data stream on the channel transmitter serial data output signal (SOUT). The transmitter status may be polled or interrupt driven.

The receiver accepts serial data bits on the channel receiver serial data input signal (SIN), converts it to parallel format, checks for a start bit, parity (if any), stop bits, and transfers the assembled character (with start, stop, parity bits removed) from the receiver buffer (or FIFO) in response to a read of the UART's receiver buffer register (URBR). The receiver status may be polled or interrupt driven.

## 12.2 External Signal Descriptions

The DUART signals are described in Table 12-1. Note that although the actual device signal names are prepended with the UART_ prefix as shown in the table, the functional (abbreviated) signal names are often used throughout this chapter.

**Table 12-1. DUART Signals—Detailed Signal Descriptions**

| Signal | I/O | Description | |
|---|---|---|---|
| UART_SIN[0:1] | I | Serial data in. Data is received on the receivers of UART0 and UART1 through the respective serial data input signal, with the least-significant bit received first. | |
| | | **State Meaning** | Asserted/Negated—Represents the data being received on the UART interface. |
| | | **Timing** | Assertion/Negation—An internal logic sample signal, *rxcnt*, uses the frequency of the baud-rate generator to sample the data on SIN. |
| UART_SOUT[0:1] | O | Serial data out. The serial data output signals for the UART0 and UART1 are set ('mark' condition) when the transmitter is disabled, idle, or operating in the local loopback mode. Data is shifted out on these signals, with the least significant bit transmitted first. | |
| | | **State Meaning** | Asserted/Negated—Represents the data being transmitted on the respective UART interface. |
| | | **Timing** | Assertion/Negation— An internal logic sample signal, *rxcnt*, uses the frequency of the baud-rate generator to update and drive the data on SOUT. |
| $\overline{\text{UART\_CTS}}$[0:1] | I | Clear to send. These active-low inputs are the clear-to-send inputs. They are connected to the respective $\overline{\text{RTS}}$ outputs of the other UART devices on the bus. They can be programmed to generate an interrupt on change-of-state of the signal. | |
| | | **State Meaning** | Asserted/Negated—Represent the clear to send condition for their respective UART. |
| | | **Timing** | Assertion/Negation—Sampled at the rising edge of every platform clock. |
| $\overline{\text{UART\_RTS}}$[0:1] | O | Request to send. $\overline{\text{UART_RTS}}$x are active-low output signals that can be programmed to be automatically negated and asserted by either the receiver or transmitter. When connected to the clear-to-send ($\overline{\text{CTS}}$) input of a transmitter, this signal can be used to control serial data flow. | |
| | | **State Meaning** | Asserted/Negated—Represents the data being transmitted on the respective UART interface. |
| | | **Timing** | Assertion/Negation—Updated and driven at the rising edge of every platform clock. |

## 12.3 Memory Map/Register Definition

Table 12-2 lists the DUART registers and their offsets. It lists the address, name, and a cross-reference to the complete description of each register. Note that the full register address is comprised of CCSRBAR together with the block base address and offset listed in Table 12-2.

There are two complete sets of DUART registers (one for each UART). The two UARTs on the device are identical, except that the registers for each UART are located at different offsets. Throughout this chapter, the registers are described by a singular acronym: for example, LCR represents the line control register for either UART0 or UART1.

The registers in each UART interface are used for configuration, control, and status. The divisor latch access bit, ULCR[DLAB], is used to access the divisor latch least- and most-significant bit registers and the alternate function register. Refer to Section 12.3.1.8, "Line Control Registers (ULCRn)," for more information on ULCR[DLAB].

All the DUART registers are one byte wide. Reads and writes to these registers must be byte-wide operations. Table 12-2 provides a register summary with references to the section and page that contains detailed information about each register. Undefined byte address spaces within offset 0x000–0xFFF are reserved.

In this table and in the register figures and field descriptions, the following access definitions apply:

- Reserved fields are always ignored for the purposes of determining access type.
- R/W, R, and W (read/write, read only, and write only) indicate that all the non-reserved fields in a register have the same access type.
- w1c indicates that all of the non-reserved fields in a register are cleared by writing ones to them.
- Mixed indicates a combination of access types.
- Special is used when no other category applies. In this case the register figure and field description table should be read carefully.

**Table 12-2. DUART Register Summary**

| DUART—Block Base Address 0x0_4000 | | | | |
|---|---|---|---|---|
| **Offset** | **Register** | **Access** | **Reset** | **Section/Page** |
| **UART0 Registers** | | | | |
| 0x500 | URBR—ULCR[DLAB] = 0 UART0 receiver buffer register | R | 0x00 | 12.3.1.1/12-5 |
| 0x500 | UTHR—ULCR[DLAB] = 0 UART0 transmitter holding register | W | 0x00 | 12.3.1.2/12-5 |
| 0x500 | UDLB—ULCR[DLAB] = 1 UART0 divisor least significant byte register | R/W | 0x00 | 12.3.1.3/12-6 |
| 0x501 | UIER—ULCR[DLAB] = 0 UART0 interrupt enable register | R/W | 0x00 | 12.3.1.4/12-7 |
| 0x501 | UDMB—ULCR[DLAB] = 1 UART0 divisor most significant byte register | R/W | 0x00 | 12.3.1.3/12-6 |
| 0x502 | UIIR—ULCR[DLAB] = 0 UART0 interrupt ID register | R | 0x01 | 12.3.1.5/12-8 |
| 0x502 | UFCR—ULCR[DLAB] = 0 UART0 FIFO control register | W | 0x00 | 12.3.1.6/12-10 |
| 0x502 | UAFR—ULCR[DLAB] = 1 UART0 alternate function register | R/W | 0x00 | 12.3.1.7/12-11 |
| 0x503 | ULCR—ULCR[DLAB] = x UART0 line control register | R/W | 0x00 | 12.3.1.8/12-11 |
| 0x504 | UMCR—ULCR[DLAB] = x UART0 modem control register | R/W | 0x00 | 12.3.1.9/12-14 |
| 0x505 | ULSR—ULCR[DLAB] = x UART0 line status register | R | 0x60 | 12.3.1.10/12-15 |
| 0x506 | UMSR—ULCR[DLAB] = x UART0 modem status register | R | 0x00 | 12.3.1.11/12-16 |
| 0x507 | USCR—ULCR[DLAB] = x UART0 scratch register | R/W | 0x00 | 12.3.1.12/12-17 |
| 0x510 | UDSR—ULCR[DLAB] = x UART0 DMA status register | R | 0x01 | 12.3.1.13/12-17 |
| **UART1 Registers** | | | | |
| 0x600–<br>0x610 | UART1 Registers[1] | | | |

[1] UART1 has the same memory-mapped registers that are described for UART0 from 0x500 to 0x510, except the offsets range from 0x600 to 0x610.

### 12.3.1 Register Descriptions

The following sections describe the UART*n* registers.

#### 12.3.1.1 Receiver Buffer Registers (URBR*n*) (ULCR[DLAB] = 0)

These registers contain the data received from the transmitter on the UART buses. In FIFO mode, when read, they return the first byte received. For FIFO status information, refer to the UDSR[RXRDY] description.

Except for the case when there is an overrun, URBR returns the data in the order it was received from the transmitter. Refer to the ULSR[OE] description, Section 12.3.1.10, "Line Status Registers (ULSRn)." Figure 12-3 shows the receiver buffer registers. Note that these registers have same offset as the UTHRs.

Figure 12-2 shows the bits in the URBRs.

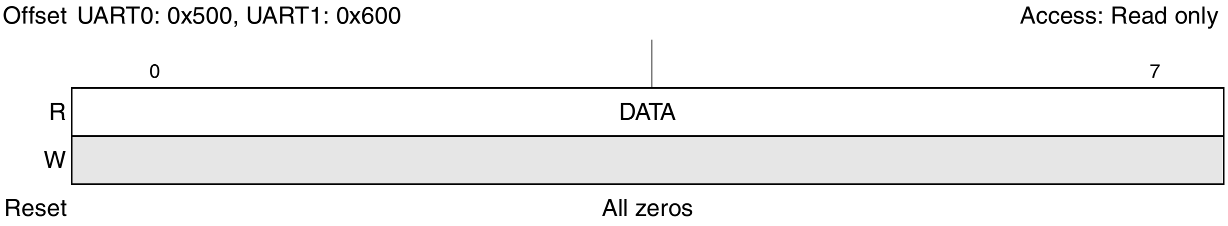


**Figure 12-2. Receiver Buffer Registers (URBR*n*)**

Table 12-3 describes the fields of URBR.

**Table 12-3. URBR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–7 | DATA | Data received from the transmitter on the UART bus (read only) |

#### 12.3.1.2 Transmitter Holding Registers (UTHR*n*) (ULCR[DLAB] = 0)

A write to these 8-bit registers causes the UART devices to transfer 5–8 data bits on the UART bus in the format set up in the ULCR (line control register). In FIFO mode, data written to UTHR is placed into the FIFO. The data written to UTHR is the data sent onto the UART bus, and the first byte written to UTHR is the first byte onto the bus. UDSR[$\overline{\text{TXRDY}}$] indicates when the FIFO is full. Refer to Table 12-20 and Table 12-21 for more details.

Figure 12-3 shows the bits in the UTHRs.

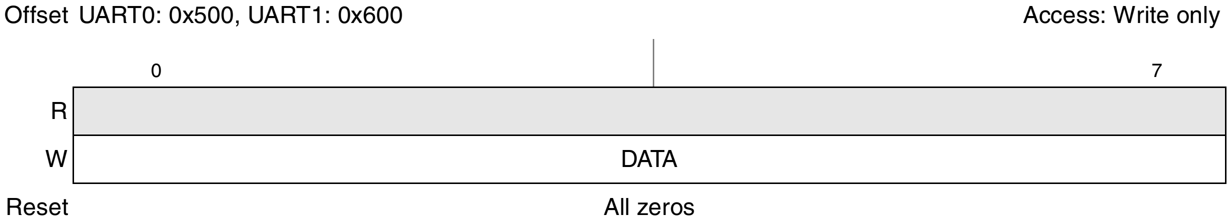


**Figure 12-3. Transmitter Holding Registers (UTHR*n*)**

Table 12-4 describes the fields of UTHR.

**Table 12-4. UTHR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–7 | DATA | Data that is written to UTHR (write only) |

### 12.3.1.3 Divisor Most and Least Significant Byte Registers (UDMB and UDLB) (ULCR[DLAB] = 1)

The divisor least significant byte register (UDLB) is concatenated with the divisor most significant byte register (UDMB) to create the divisor used to divide the input clock into the DUART. The output frequency of the baud generator is 16 times the baud rate; therefore the desired baud rate = platform clock frequency/(16 × [UDMB||UDLB]). Equivalently, [UDMB||UDLB:0b0000] = platform clock frequency/desired baud rate. Baud rates that can be generated by specific input clock frequencies are shown in Table 12-7.

Figure 12-4 shows the bits in the UDMBs.

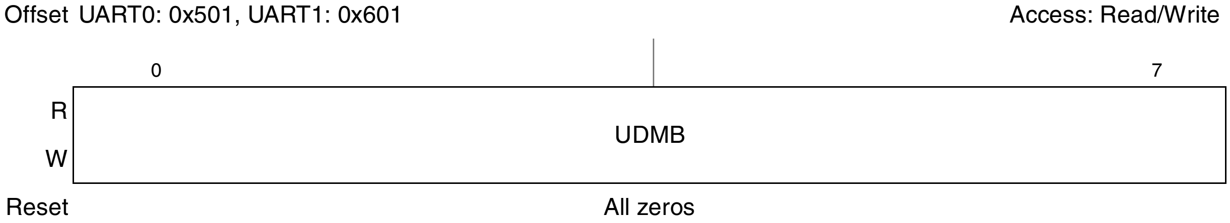


**Figure 12-4. Divisor Most Significant Byte Registers (UDMB0, UDMB1)**

Table 12-5 describes the fields of UDMB registers.

**Table 12-5. UDMB Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–7 | UDMB | Divisor most significant byte |

Figure 12-5 shows the bits in the UDLBs.

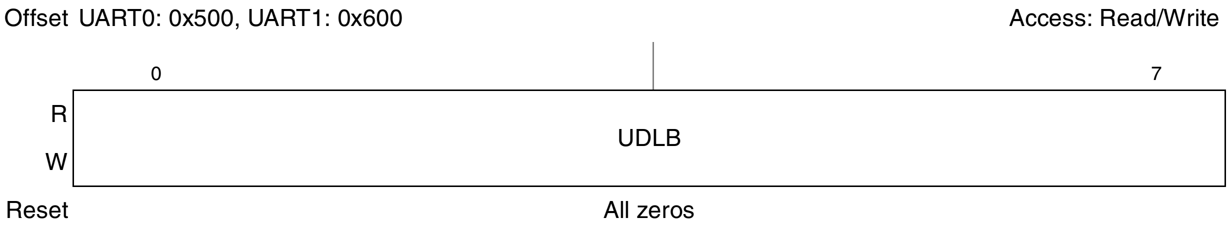


**Figure 12-5. Divisor Least Significant Byte Registers (UDLB*n*)**

Table 12-6 describes the fields of UDLB registers.

**Table 12-6. UDLB Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–7 | UDLB | Divisor least significant byte. This is concatenated with UDMB. |

Table 12-7 shows examples of baud rate generation based on common input clock frequencies. Many other target baud rates are also possible. Note that because only integer values can be used as divisors, the actual baud rate differs slightly from the desired (target) baud rate; for this reason, both target and actual baud rates are given, along with the percentage of error.

**Table 12-7. Baud Rate Examples**

| Target Baud Rate (Decimal) | Divisor | | Platform Clock (CCB) Frequency (MHz) | Actual Baud Rate (Decimal) | Percent Error (Decimal) |
|---|---|---|---|---|---|
| | Decimal | Hex | | | |
| 9,600 | 2170 | 87A | 333 | 9600.61444 | 0.0064 |
| 19,200 | 1085 | 43D | 333 | 19,201.22888 | 0.0064 |
| 38,400 | 543 | 21F | 333 | 38,367.09638 | 0.0858 |
| 57,600 | 362 | 16A | 333 | 57,550.64457 | 0.0857 |
| 115,200 | 181 | B5 | 333 | 115,101.28913 | 0.0857 |
| 230,400 | 90 | 5A | 333 | 231,481.48148 | 0.4694 |

### 12.3.1.4 Interrupt Enable Register (UIER) (ULCR[DLAB] = 0)

The UIER gives the user the ability to mask specific UART interrupts to the programmable interrupt controller (PIC).

Figure 12-6 shows the bits in the UIER.

Offset UART0: 0x501, UART1: 0x601 Access: Read/Write

| | 0–3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|
| R<br>W | — | EMSI | ERLSI | ETHREI | ERDAI |
| Reset | All zeros | | | | |

**Figure 12-6. Interrupt Enable Register (UIER)**

Table 12-8 describes the fields of UIER.

**Table 12-8. UIER Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–3 | — | Reserved. |
| 4 | EMSI | Enable modem status interrupt.<br>0 Mask interrupts caused by UMSR[DCTS] being set<br>1 Enable and assert interrupts when the clear-to-send bit in the UART modem status register (UMSR) changes state |
| 5 | ERLSI | Enable receiver line status interrupt.<br>0 Mask interrupts when ULSR's overrun, parity error, framing error or break interrupt bits are set<br>1 Enable and assert interrupts when ULSR's overrun, parity error, framing error or break interrupt bits are set |
| 6 | ETHREI | Enable transmitter holding register empty interrupt.<br>0 Mask interrupt when ULSR[THRE] is set<br>1 Enable and assert interrupts when ULSR[THRE] is set |
| 7 | ERDAI | Enable received data available interrupt.<br>0 Mask interrupt when new receive data is available or receive data time out has occurred<br>1 Enable and assert interrupts when a new data character is received from the external device and/or a time-out interrupt occurs in the FIFO mode |

#### 12.3.1.5 Interrupt ID Registers (UIIR*n*) (ULCR[DLAB] = 0)

The UIIRs indicate when an interrupt is pending from the corresponding UART and what type of interrupt is active. They also indicate if the FIFOs are enabled.

The DUART prioritizes interrupts into four levels and records these in the corresponding UIIR. The four levels of interrupt conditions in order of priority are:

1. Receiver line status
2. Received data ready/character time-out
3. Transmitter holding register empty
4. Modem status

See Table 12-10 for more details.

When the UIIR is read, the associated DUART serial channel freezes all interrupts and indicates the highest priority pending interrupt. While this read transaction is occurring, the associated DUART serial channel records new interrupts, but does not change the contents of UIIR until the read access is complete.

Figure 12-7 shows the bits in the UIIR.

Offset UART0: 0x502, UART1: 0x602 Access: Read only

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| R | FE | | — | | IID3 | IID2 | IID1 | IID0 |
| W | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

**Figure 12-7. Interrupt ID Registers (UIIR)**

Table 12-9 describes the fields of the UIIR.

**Table 12-9. UIIR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–1 | FE | FIFOs enabled. Reflects the setting of UFCR[FEN] |
| 2–3 | — | Reserved |
| 4 | IID3 | Interrupt ID bits identify the highest priority interrupt that is pending as indicated in Table 12-10. IID3 is set along with IID2 only when a timeout interrupt is pending for FIFO mode. |
| 5–6 | IID2–1 | Interrupt ID bits identify the highest priority interrupt that is pending as indicated in Table 12-10. |
| 7 | IID0 | IID0 indicates when an interrupt is pending.<br>0 The UART has an active interrupt ready to be serviced.<br>1 No interrupt is pending. |

The bits contained in the UIIR registers are described in Table 12-10.

**Table 12-10. UIIR IID Bits Summary**

| IID Bits IID[3–0] | Priority Level | Interrupt Type | Interrupt Description | How To Reset Interrupt |
|---|---|---|---|---|
| 0b0001 | — | — | — | — |
| 0b0110 | Highest | Receiver line status | Overrun error, parity error, framing error, or break interrupt | Read the line status register. |
| 0b0100 | Second | Received data available | Receiver data available or trigger level reached in FIFO mode | Read the receiver buffer register or interrupt is automatically reset if the number of bytes in the receiver FIFO drops below the trigger level. |
| 0b1100 | Second | Character time-out | No characters have been removed from or input to the receiver FIFO during the last 4 character times and there is at least one character in the receiver FIFO during this time. | Read the receiver buffer register. |

Table 12-10. UIIR IID Bits Summary (continued)

| IID Bits IID[3–0] | Priority Level | Interrupt Type | Interrupt Description | How To Reset Interrupt |
|---|---|---|---|---|
| 0b0010 | Third | UTHR empty | Transmitter holding register is empty | Read the UIIR or write to the UTHR. |
| 0b0000 | Fourth | Modem status | $\overline{\text{CTS}}$ input value changed since last read of UMSR | Read the UMSR. |

### 12.3.1.6 FIFO Control Registers (UFCR*n*) (ULCR[DLAB] = 0)

The UFCR, a write-only register, is used to enable and clear the receiver and transmitter FIFOs, set a receiver FIFO trigger level to control the received data available interrupt, and select the type of DMA signaling.

When the UFCR bits are written, the FIFO enable bit must also be set or else the UFCR bits are not programmed. When changing from FIFO mode to 16450 mode (non-FIFO mode) and vice versa, data is automatically cleared from the FIFOs.

After all the bytes in the receiver FIFO are cleared, the receiver internal shift register is not cleared. Similarly, the bytes are cleared in the transmitter FIFO, but the transmitter internal shift register is not cleared. Both TFR and RFR are self-clearing bits.

Figure 12-8 shows the bits in the UFCRs.

Offset UART0: 0x502, UART1: 0x602 Access: Write only

|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| R |  |  | — |  |  |  |  |  |
| W | RTL |  |  |  | DMS | TFR | RFR | FEN |
| Reset | All zeros |  |  |  |  |  |  |  |

Figure 12-8. FIFO Control Registers (UFCR*n*)

Table 12-11 describes the fields of the UFCRs.

Table 12-11. UFCR Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–1 | RTL | Receiver trigger level. A received data available interrupt occurs when UIER[ERDAI] is set and the number of bytes in the receiver FIFO equals the designated interrupt trigger level as follows:<br>00 1 byte<br>01 4 bytes<br>10 8 bytes<br>11 14 bytes |
| 2–3 | — | Reserved |
| 4 | DMS | DMA mode select. See Section 12.4.5.2, “DMA Mode Select,” for more information.<br>0 UDSR[RXRDY] and UDSR[TXRDY] bits are in mode 0.<br>1 UDSR[RXRDY] and UDSR[TXRDY] bits are in mode 1 if UFCR[FEN] = 1. |

Table 12-11. UFCR Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 5 | TFR | Transmitter FIFO reset<br>0 No action<br>1 Clears all bytes in the transmitter FIFO and resets the FIFO counter/pointer to 0 |
| 6 | RFR | Receiver FIFO reset<br>0 No action<br>1 Clears all bytes in the receiver FIFO and resets the FIFO counter/pointer to 0 |
| 7 | FEN | FIFO enable<br>0 FIFOs are disabled and cleared<br>1 Enables the transmitter and receiver FIFOs |

### 12.3.1.7 Alternate Function Registers (UAFR*n*) (ULCR[DLAB] = 1)

The UAFRs give software the ability to gate off the baud clock and write to both UART0/UART1 registers simultaneously with the same write operation.

Figure 12-9 shows the bits in the UAFRs.

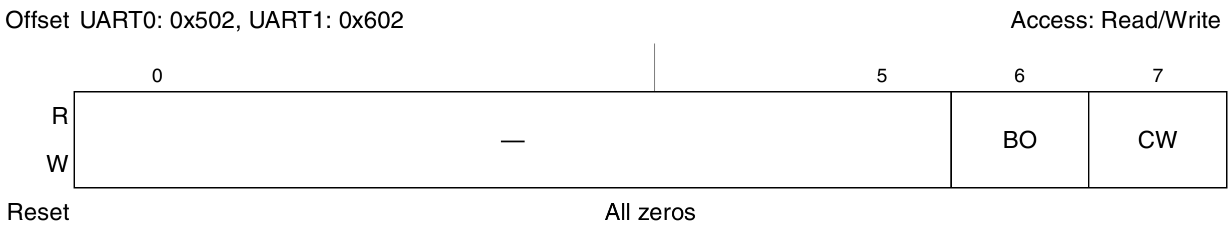


Figure 12-9. Alternate Function Register (UAFR)

Table 12-12 describes the fields of the UAFRs.

Table 12-12. UAFR Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–5 | — | Reserved. |
| 6 | BO | Baud clock select.<br>0 The baud clock is not gated off.<br>1 The baud clock is gated off. |
| 7 | CW | Concurrent write enable.<br>0 Disables writing to both UART0 and UART1<br>1 Enables concurrent writes to corresponding UART registers. A write to a register in UART0 is also a write to the corresponding register in UART1 and vice versa. The user needs to ensure that the LCR[DLAB] of both UARTs are in the same state before executing a concurrent write to register addresses 0x*n*00, 0x*n*01 and 0x*n*02, where *n* is the offset of the corresponding UART. |

### 12.3.1.8 Line Control Registers (ULCR*n*)

The ULCRs specify the data format for the UART bus and set the divisor latch access bit ULCR[DLAB], which controls the ability to access the divisor latch least and most significant bit registers and the alternate function register.

After initializing the ULCR, the software should not re-write the ULCR when valid transfers on the UART bus are active. The software should not re-write the ULCR until the last STOP bit has been received and there are no new characters being transferred on the bus.

The stick parity bit, ULCR[SP], assigns a set parity value for the parity bit time slot sent on the UART bus. The set value is defined as mark parity (logic 1) or space parity (logic 0). ULCR[PEN] and ULCR[EPS] help determine the set parity value. See Table 12-14 for more information. ULCR[NSTB], defines the number of STOP bits to be sent at the end of the data transfer. The receiver only checks the first STOP bit, regardless of the number of STOP bits selected. The word length select bits (1 and 0) define the number of data bits that are transmitted or received as a serial character. The word length does not include START, parity, and STOP bits.

Figure 12-10 shows the bits in the ULCRs.

Offset UART0: 0x503, UART1 0x603 Access: Read/Write

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| R<br>W | DLAB | SB | SP | EPS | PEN | NSTB | WLS | |
| Reset | All zeros | | | | | | | |

**Figure 12-10. Line Control Register (ULCR)**

Table 12-13 describes the fields of the ULCRs.

**Table 12-13. ULCR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | DLAB | Divisor latch access bit.<br>0 Access to all registers except UDLB, UAFR, and UDMB<br>1 Ability to access divisor latch least and most significant byte registers and alternate function register (UAFR) |
| 1 | SB | Set break.<br>0 Send normal UTHR data onto the serial output (SOUT) signal<br>1 Force logic 0 to be on the SOUT signal. Data in the UTHR is not affected |
| 2 | SP | Stick parity.<br>0 Stick parity is disabled.<br>1 If PEN = 1 and EPS = 1, space parity is selected. And if PEN = 1 and EPS = 0, mark parity is selected. |
| 3 | EPS | Even parity select. See Table 12-14 for more information.<br>0 If PEN = 1 and SP = 0, odd parity is selected.<br>1 If PEN = 1 and SP = 0, even parity is selected. |
| 4 | PEN | Parity enable.<br>0 No parity generation and checking<br>1 Generate parity bit as a transmitter, and check parity as a receiver |

**Table 12-13. ULCR Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 5 | NTSB | Number of STOP bits.<br>0 One STOP bit is generated in the transmitted data.<br>1 When a 5-bit data length is selected, 1∫ STOP bits are generated. When either a 6-, 7-, or 8-bit word length is selected, two STOP bits are generated. |
| 6–7 | WLS | Word length select. Number of bits that comprise the character length. The word length select values are as follows:<br>00 5 bits<br>01 6 bits<br>10 7 bits<br>11 8 bits |

**Table 12-14. Parity Selection Using ULCR[PEN], ULCR[SP], and ULCR[EPS]**

| PEN | SP | EPS | Parity Selected |
|---|---|---|---|
| 0 | 0 | 0 | No parity |
| 0 | 0 | 1 | No parity |
| 0 | 1 | 0 | No parity |
| 0 | 1 | 1 | No parity |
| 1 | 0 | 0 | Odd parity |
| 1 | 0 | 1 | Even parity |
| 1 | 1 | 0 | Mark parity |
| 1 | 1 | 1 | Space parity |

### 12.3.1.9 Modem Control Registers (UMCR*n*)

The UMCRs control the interface with the external peripheral device on the UART bus.

Figure 12-11 shows the bits in the UMCRs

Offset UART0: 0x504, UART1: 0x604 Access: Read/Write

| | 0 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| R<br>W | — | | LOOP | — | | RTS | — |
| Reset | All zeros | | | | | | |

**Figure 12-11. Modem Control Register (UMCR)**

Table 12-15 describes the fields of UMCRs.

**Table 12-15. UMCR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–2 | — | Reserved. |
| 3 | LOOP | Local loopback mode.<br>0 Normal operation<br>1 Functionally, the data written to UTHR can be read from URBR of the same UART, and UMCR[RTS] is tied to UMSR[CTS]. |
| 4–5 | — | Reserved. |
| 6 | RTS | Ready to send.<br>0 Negates corresponding $\overline{\text{UART\_RTS}}$ output<br>1 Assert corresponding $\overline{\text{UART\_RTS}}$ output. Informs external modem or peripheral that the UART is ready for sending/receiving data |
| 7 | — | Reserved. |

### 12.3.1.10 Line Status Registers (ULSR*n*)

The ULSRs are read-only registers that monitor the status of the data transfer on the UART buses. To isolate the status bits from the proper character received through the UART bus, software should read the ULSR and then the URBR.

Figure 12-12 shows the bits in the ULSRs.

Offset UART0: 0x505, UART1: 0x605 Access: Read only

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| R | RFE | TEMT | THRE | BI | FE | PE | OE | DR |
| W | | | | | | | | |
| Reset | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |

**Figure 12-12. Line Status Register (ULSR)**

Table 12-16 describes the fields of the ULSRs.

**Table 12-16. ULSR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | RFE | Receiver FIFO error.<br>0 This bit is cleared when there are no errors in the receiver FIFO or on a read of the ULSR with no remaining receiver FIFO errors.<br>1 Set to one when one of the characters in the receiver FIFO encounters an error (framing, parity, or break interrupt) |
| 1 | TEMT | Transmitter empty.<br>0 Either or both the UTHR or the internal transmitter shift register has a data character. In FIFO mode, a data character is in the transmitter FIFO or the internal transmitter shift register.<br>1 Both the UTHR and the internal transmitter shift register are empty. In FIFO mode, both the transmitter FIFO and the internal transmitter shift register are empty. |
| 2 | THRE | Transmitter holding register empty.<br>0 The UTHR is not empty.<br>1 A data character has transferred from the UTHR into the internal transmitter shift register. In FIFO mode, the transmitter FIFO contains no data character. |
| 3 | BI | Break interrupt.<br>0 This bit is cleared when the ULSR is read or when a valid data transfer is detected (that is, STOP bit is received).<br>1 Received data of logic 0 for more than START bit + Data bits + Parity bit + one STOP bits length of time. A new character is not loaded until SIN returns to the mark state (logic 1) and a valid START is detected. In FIFO mode, a zero character is encountered in the FIFO (the zero character is at the top of the FIFO). In FIFO mode, only one zero character is stored. |
| 4 | FE | Framing error.<br>0 This bit is cleared when ULSR is read or when a new character is loaded into the URBR from the receiver shift register.<br>1 Invalid STOP bit for receive data (only the first STOP bit is checked). In FIFO mode, this bit is set when the character that detected a framing error is encountered in the FIFO (that is the character at the top of the FIFO). An attempt to resynchronize occurs after a framing error. The UART assumes that the framing error (due to a logic 0 being read when a logic 1 (STOP) was expected) was due to a STOP bit overlapping with the next START bit, so it assumes this logic 0 sample is a true START bit and then receives the following new data. |

Table 12-16. ULSR Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 5 | PE | Parity error.<br>0 This bit is cleared when ULSR is read or when a new character is loaded into the URBR.<br>1 Unexpected parity value encountered when receiving data. In FIFO mode, the character with the error is at the top of the FIFO. |
| 6 | OE | Overrun error.<br>0 This bit is cleared when ULSR is read.<br>1 Before the URBR is read, the URBR was overwritten with a new character. The old character is loss. In FIFO mode, the receiver FIFO is full (regardless of the receiver FIFO trigger level setting) and a new character has been received into the internal receiver shift register. The old character was overwritten by the new character. Data in the receiver FIFO was not overwritten. |
| 7 | DR | Data ready.<br>0 This bit is cleared when URBR is read or when all of the data in the receiver FIFO is read.<br>1 A character has been received in the URBR or the receiver FIFO. |

### 12.3.1.11 Modem Status Registers (UMSR*n*)

The UMSRs track the status of the modem (or external peripheral device) clear to send ($\overline{\text{CTS}}$) signal for the corresponding UART.

Figure 12-13 shows the bits in the UMSRs.

Offset UART0: 0x506, UART10: 0x606 Access: Read only

|  | 0 | 2 | 3 | 4 | 6 | 7 |
|---|---|---|---|---|---|---|
| R | — |  | CTS | — |  | DCTS |
| W |  |  |  |  |  |  |
| Reset | All zeros |  |  |  |  |  |

Figure 12-13. Modem Status Register (UMSR)

Table 12-17 describes the fields of the UMSRs.

Table 12-17. UMSR Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–2 | — | Reserved. |
| 3 | CTS | Clear to send. Represents the inverted value of the $\overline{\text{CTS}}$ input pin from the external peripheral device<br>0 Corresponding $\overline{\text{CTS}}$*n* is negated<br>1 Corresponding $\overline{\text{CTS}}$*n* is asserted. The modem or peripheral device is ready for data transfers. |
| 4–6 | — | Reserved. |
| 7 | DCTS | Clear to send.<br>0 No change on the corresponding $\overline{\text{CTS}}$*n* signal since the last read of UMSR[CTS]<br>1 The $\overline{\text{CTS}}$*n* value has changed, since the last read of UMSR[CTS]. Causes an interrupt if UIER[EMSI] is set to detect this condition |

### 12.3.1.12 Scratch Registers (USCR*n*)

The USCR registers are for debugging software or the DUART hardware. The USCRs do not affect the operation of the DUART.

Figure 12-14 shows the bits in USCRs.

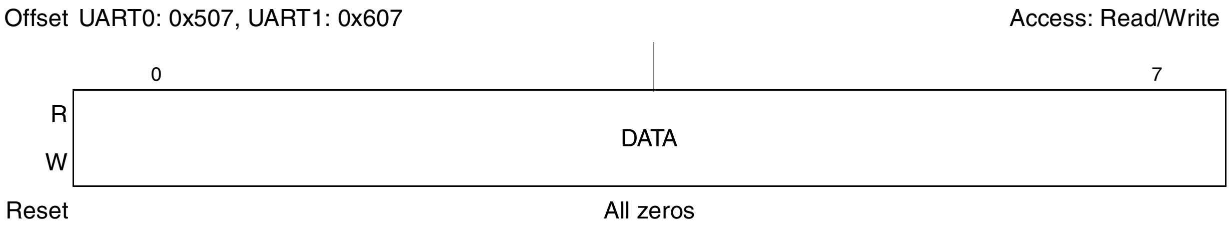


**Figure 12-14. Scratch Register (USCR)**

Table 12-18 describes the fields of the USCRs.

**Table 12-18. USCR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–7 | DATA | Data |

### 12.3.1.13 DMA Status Registers (UDSR*n*)

The DMA status registers (UDSRs) are read-only registers that return transmitter and receiver FIFO status. UDSRs also provide the ability to assist DMA data operations to and from the FIFOs.

Figure 12-15 shows the bits in UDSRs.

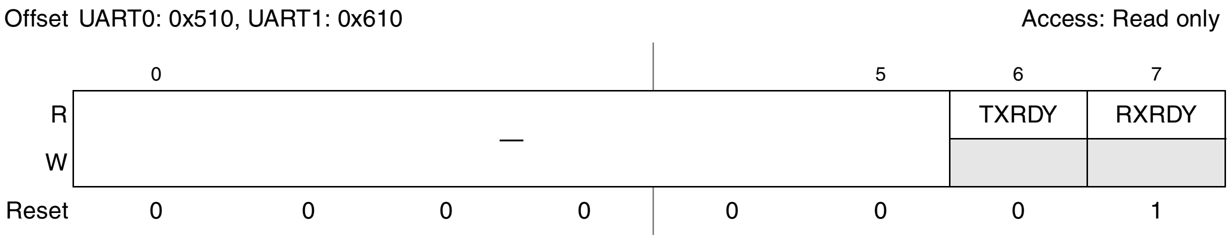


**Figure 12-15. DMA Status Register (UDSR)**

Table 12-19 describes the fields of the UDSRs.

**Table 12-19. UDSR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–5 | — | Reserved |

Table 12-19. UDSR Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 6 | TXRDY | Transmitter ready. This read-only bit reflects the status of the transmitter FIFO or the UTHR. The status depends on the DMA mode selected, which is determined by the DMS and FEN bits in the UFCR.<br>0 The bit is cleared, as shown in Table 12-21.<br>1 This bit is set, as shown in Table 12-20. |
| 7 | RXRDY | Receiver ready. This read-only bit reflects the status of the receiver FIFO or URBR. The status depends on the DMA mode selected, which is determined by the DMS and FEN bits in the UFCR.<br>0 The bit is cleared, as shown in Table 12-23.<br>1 This bit is set, as shown in Table 12-22. |

Table 12-20. UDSR[TXRDY] Set Conditions

| DMS | FEN | DMA Mode | Meaning |
|---|---|---|---|
| 0 | 0 | 0 | TXRDY is set after the first character is loaded into the transmitter FIFO or UTHR. |
| 0 | 1 | 0 | |
| 1 | 0 | 0 | |
| 1 | 1 | 1 | TXRDY is set when the transmitter FIFO is full. |

Table 12-21. UDSR[TXRDY] Cleared Conditions

| DMS | FEN | DMA Mode | Meaning |
|---|---|---|---|
| 0 | 0 | 0 | TXRDY is cleared when there are no characters in the transmitter FIFO or UTHR. |
| 0 | 1 | 0 | |
| 1 | 0 | 0 | |
| 1 | 1 | 1 | TXRDY is cleared when there are no characters in the transmitter FIFO or UTHR. TXRDY remains clear when the transmitter FIFO is not yet full. |

Table 12-22. UDSR[RXRDY] Set Conditions

| DMS | FEN | DMA Mode | Meaning |
|---|---|---|---|
| 0 | 0 | 0 | RXRDY is set when there are no characters in the receiver FIFO or URBR. |
| 0 | 1 | 0 | |
| 1 | 0 | 0 | |
| 1 | 1 | 1 | RXRDY is set when the trigger level has not been reached and there has been no time out. |

Table 12-23. UDSR[RXRDY] Cleared Conditions

| DMS | FEN | DMA Mode | Meaning |
|---|---|---|---|
| 0 | 0 | 0 | RXRDY is cleared when there is at least one character in the receiver FIFO or URBR. |
| 0 | 1 | 0 | |
| 1 | 0 | 0 | |
| 1 | 1 | 1 | RXRDY is cleared when the trigger level or a time-out has been reached. RXRDY remains cleared until the receiver FIFO is empty. |

## 12.4 Functional Description

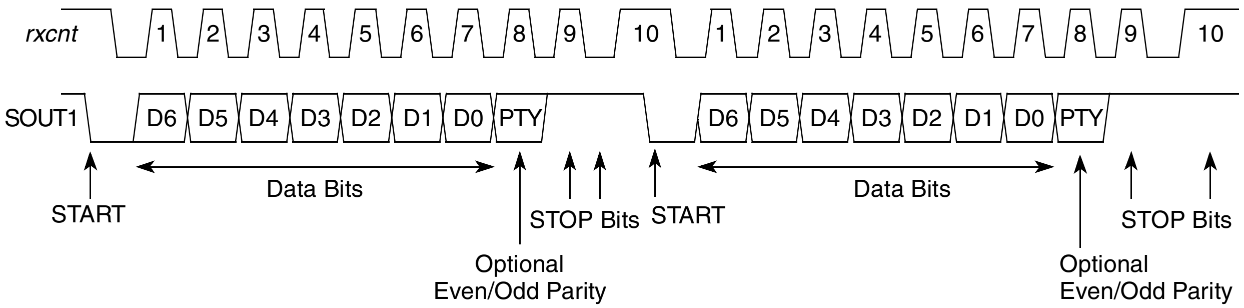
The communication channel provides a full-duplex asynchronous receiver and transmitter using an operating frequency derived from the platform clock signal.

The transmitter accepts parallel data with a write access to the transmitter holding register (UTHR). In FIFO mode, the data is placed directly into an internal transmitter shift register, or into the transmitter FIFO—see Section 12.4.5, “FIFO Mode.” The transmitting registers convert the data to a serial bit stream, by inserting the appropriate START, STOP, and optional parity bits. Finally, the registers output a composite serial data stream on the channel transmitter serial data output (SOUT). The transmitter status may be polled or interrupt-driven.

The receiver accepts serial data on the channel receiver serial data input (SIN), converts the data into parallel format, and checks for START, STOP, and parity bits. In FIFO mode, the receiver removes the START, STOP, and parity bits and then transfers the assembled character from the receiver buffer, or receiver FIFO. This transfer occurs in response to a read of the UART receiver buffer register (URBR). The receiver status may be polled or interrupt driven.

### 12.4.1 Serial Interface

The UART bus is a serial, full-duplex, point-to-point bus as shown in Figure 12-16. Therefore, only two devices are attached to the same signals and there is no need for address or arbitration bus cycles.



Two 7-bit data transmissions with parity and 2-bit STOP transactions

Figure 12-16. UART Bus Interface Transaction Protocol Example

A standard UART bus transfer is composed of either three or four parts:

- START bit
- Data transfer bits (least-significant bit is first data bit on the bus)
- Parity bit (optional)
- STOP bits

An internal logic sample signal, *rxcnt*, uses the frequency of the baud-rate generator to drive the bits on SOUT.

The following sections describe the four components of the serial interface, the baud-rate generator, local loopback mode, different errors, and FIFO mode.

#### 12.4.1.1 START Bit

A write to the transmitter holding register (UTHR) generates a START bit on the SOUT signal. Figure 12-16 shows that the START bit is defined as a logic 0. The START bit denotes the beginning of a new data transfer which is limited to the bit length programmed in the UART line control register (ULCR).When the bus is idle, SOUT is high.

#### 12.4.1.2 Data Transfer

Each data transfer contains 5–8 bits of data. The ULCR data bit length for the transmitter and receiver UART devices must agree before a transfer begins; otherwise, a parity or framing error may occur. A transfer begins when UTHR is written. At that time a START bit is generated followed by 5–8 of the data bits previously written to the UTHR. The data bits are driven from the least significant to the most significant bits. After the parity and STOP bits, a new data transfer can begin if new data is written to the UTHR.

#### 12.4.1.3 Parity Bit

The user has the option of using even, odd, no parity, or stick parity (see Section 12.3.1.8, “Line Control Registers (ULCRn).” Both the receiver and transmitter parity definition must agree before attempting to transfer data. When receiving data a parity error can occur if an unexpected parity value is detected. (See Section 12.3.1.10, “Line Status Registers (ULSRn).”)

#### 12.4.1.4 STOP Bit

The transmitter device ends the write transfer by generating a STOP bit. The STOP bit is always high. The user can program the length of the STOP bit(s) in the ULCR. Both the receiver and transmitter STOP bit length must agree before attempting to transfer data. A framing error can occur if an invalid STOP bit is detected.

### 12.4.2 Baud-Rate Generator Logic

Each UART contains an independent programmable baud-rate generator, that is capable of taking the platform clock input and dividing the input by any divisor from 1 to $2^{16} – 1$.

The baud rate is defined as the number of bits per second that can be sent over the UART bus. The formula for calculating baud rate is as follows:

$$\text{Baud rate} = (1/16) \times (\text{platform clock frequency/divisor value})$$

Therefore, the output frequency of the baud-rate generator is 16 times the baud rate.

The divisor value is determined by the following two 8-bit registers to form a 16-bit binary number:

- UART divisor most significant byte register (UDMB)
- UART divisor least significant byte register (UDLB)

Upon loading either of the divisor latches, a 16-bit baud-rate counter is loaded.

The divisor latches must be loaded during initialization to ensure proper operation of the baud-rate generator. Both UART devices on the same bus must be programmed for the same baud-rate before starting a transfer.

The baud clock can be passed to the performance monitor by enabling the UAFR[BO] bit. This can be used to determine baud rate errors.

### 12.4.3 Local Loopback Mode

Local loopback mode is provided for diagnostic testing. The data written to UTHR can be read from the receiver buffer register (URBR) of the same UART. In this mode, the modem control register UMCR[RTS] is internally tied to the modem status register UMSR[CTS]. The transmitter SOUT is set to a logic 1 and the receiver SIN is disconnected. The output of the transmitter shift register is looped back into the receiver shift register input. The $\overline{\text{CTS}}$ (input signal) is disconnected, $\overline{\text{RTS}}$ is internally connected to $\overline{\text{CTS}}$, and the $\overline{\text{RTS}}$ (output signal) becomes inactive. In this diagnostic mode, data that is transmitted is immediately received. In local loopback mode the transmit and receive data paths of the DUART can be verified. Note that in local loopback mode, the transmit/receive interrupts are fully operational and can be controlled by the interrupt enable register (UIER).

### 12.4.4 Errors

The following sections describe framing, parity, and overrun errors which may occur while data is transferred on the UART bus. Each of the error bits are usually cleared, as described below, when the line status register (ULSR) is read.

#### 12.4.4.1 Framing Error

When an invalid STOP bit is detected, a framing error occurs and ULSR[FE] is set. Note that only the first STOP bit is checked. In FIFO mode, ULSR[FE] is set when the character at the top of the FIFO detects a framing error. An attempt to re-synchronize occurs after a framing error. The UART assumes that the framing error (due to a logic 0 being read when a logic 1 (STOP) was expected) was due to a STOP bit overlapping with the next START bit. ULSR[FE] is cleared when ULSR is read or when a new character is loaded into the URBR from the receiver shift register.

#### 12.4.4.2 Parity Error

A parity error occurs, and ULSR[PE] is set, when unexpected parity values are encountered while receiving data. In FIFO mode, ULSR[PE] is set when the character with the error is at the top of the FIFO. ULSR[PE] is cleared when ULSR is read or when a new character is loaded into the URBR.

#### 12.4.4.3 Overrun Error

When a new (overwriting character) STOP bit is detected and the old character is lost, an overrun error occurs and ULSR[OE] is set. In FIFO mode, ULSR[OE] is set after the receiver FIFO is full (despite the receiver FIFO trigger level setting) and a new character has been received into the internal receiver shift register. Data in the FIFO is not overwritten; only the shift register data is overwritten. Therefore, the interrupt occurs immediately. ULSR[OE] is cleared when ULSR is read.

### 12.4.5 FIFO Mode

The UARTs use an alternate mode (FIFO mode) to relieve the processor core from excessive software overhead. The FIFO control register (UFCR) is used to enable and clear the receiver and transmitter FIFOs and set the FIFO receiver trigger level UFCR[RTL] to control the received data available interrupt UIER[ERDAI].

The UFCR also selects the type of DMA signaling. The UDSR[RXRDY] indicates the status of the receiver FIFO. The DMA status registers (UDSR[TXRDY]) indicate when the transmitter FIFO is full. When in FIFO mode, data written to UTHR is placed into the transmitter FIFO. The first byte written to UTHR is the first byte onto the UART bus.

#### 12.4.5.1 FIFO Interrupts

In FIFO mode, the UIER[ERDAI] is set when a time-out interrupt occurs. When a receive data time-out occurs there is a maskable interrupt condition (through UIER[ERDAI]). See Section 12.3.1.4, “Interrupt Enable Register (UIER) (ULCR[DLAB] = 0),” for more details on interrupt enables.

The interrupt ID register (UIIR) indicates if the FIFOs are enabled. Interrupt ID3 UIIR[IID3] bit is only set for FIFO mode interrupts. The character time-out interrupt occurs when no characters have been removed from or input to the receiver FIFO during the last four character times and there is at least one character in the receiver FIFO during this time. The character time-out interrupt (controlled by UIIR[IID*n*]) is cleared when the URBR is read. See Section 12.3.1.5, “Interrupt ID Registers (UIIRn) (ULCR[DLAB] = 0),” for more information.

The UIIR[FE] bits indicate if FIFO mode is enabled.

#### 12.4.5.2 DMA Mode Select

The UDSR[RXRDY] bit reflects the status of the receiver FIFO or URBR. In mode 0 (UFCR[DMS] is cleared), UDSR[RXRDY] is cleared when there is at least one character in the receiver FIFO or URBR and it is set when there are no more characters in the receiver FIFO or URBR. This occurs regardless of the setting of the UFCR[FEN] bit. In mode 1 (UFCR[DMS] and UFCR[FEN] are set), UDSR[RXRDY] is cleared when the trigger level or a time-out has been reached and it is set when there are no more characters in the receiver FIFO.

The UDSR[TXRDY] bit reflects the status of the transmitter FIFO or UTHR. In mode 0 (UFCR[DMS] is cleared), UDSR[TXRDY] is cleared when there are no characters in the transmitter FIFO or UTHR and it is set after the first character is loaded into the transmitter FIFO or UTHR. This occurs regardless of the setting of the UFCR[FEN] bit. In mode 1 (UFCR[DMS] and UFCR[FEN] are set), UDSR[TXRDY] is cleared when there are no characters in the transmitter FIFO or UTHR and it is set when the transmitter FIFO is full.

See Section 12.3.1.13, “DMA Status Registers (UDSRn),” for a complete description of the USDR[RXRDY] and USDR[TXRDY] bits.

### 12.4.5.3 Interrupt Control Logic

An interrupt is active when DUART interrupt ID register bit 7 (UIIR[IID0]), is cleared. The interrupt enable register (UIER) is used to mask specific interrupt types. For more details refer to the description of UIER in Section 12.3.1.4, “Interrupt Enable Register (UIER) (ULCR[DLAB] = 0).”

When the interrupts are disabled in UIER, polling software cannot use UIIR[IID0] to determine whether the UART is ready for service. The software must monitor the appropriate bits in the line status (ULSR) and/or the modem status (UMSR) registers. UIIR[IID0] can be used for polling if the interrupts are enabled in UIER.

## 12.5 DUART Initialization/Application Information

The following requirements must be met for DUART accesses:

- All DUART registers must be mapped to a cache-inhibited and guarded area. (That is, the WIMG setting in the MMU needs to be 0b01X1.)
- All DUART registers are 1 byte wide. Reads and writes to these registers must be byte-wide operations.

A system reset puts the DUART registers to a default state. Before the interface can transfer serial data, the following initialization steps are recommended:

1. Update the programmable interrupt controller (PIC) DUART channel interrupt vector source registers.
2. Set data attributes and control bits in the ULCR, UFCR, UAFR, UMCR, UDLB, and UDMB.
3. Set the data attributes and control bits of the external modem or peripheral device.
4. Set the interrupt enable register (UIER).
5. To start a write transfer, write to the UTHR.
6. Poll UIIR if the interrupts generated by the DUART are masked.

# Chapter 13
# Enhanced Local Bus Controller

This chapter describes the enhanced local bus controller (eLBC) block. It describes the external signals and the memory-mapped registers as well as a functional description of the general-purpose chip-select machine (GPCM), NAND Flash control machine (FCM), and user-programmable machines (UPMs) of the eLBC. Finally, it includes an initialization and applications information section with many specific examples of its use.

## 13.1 Introduction

Figure 13-1 is a functional block diagram of the eLBC, which supports three interfaces: GPCM, FCM, and UPM controllers.

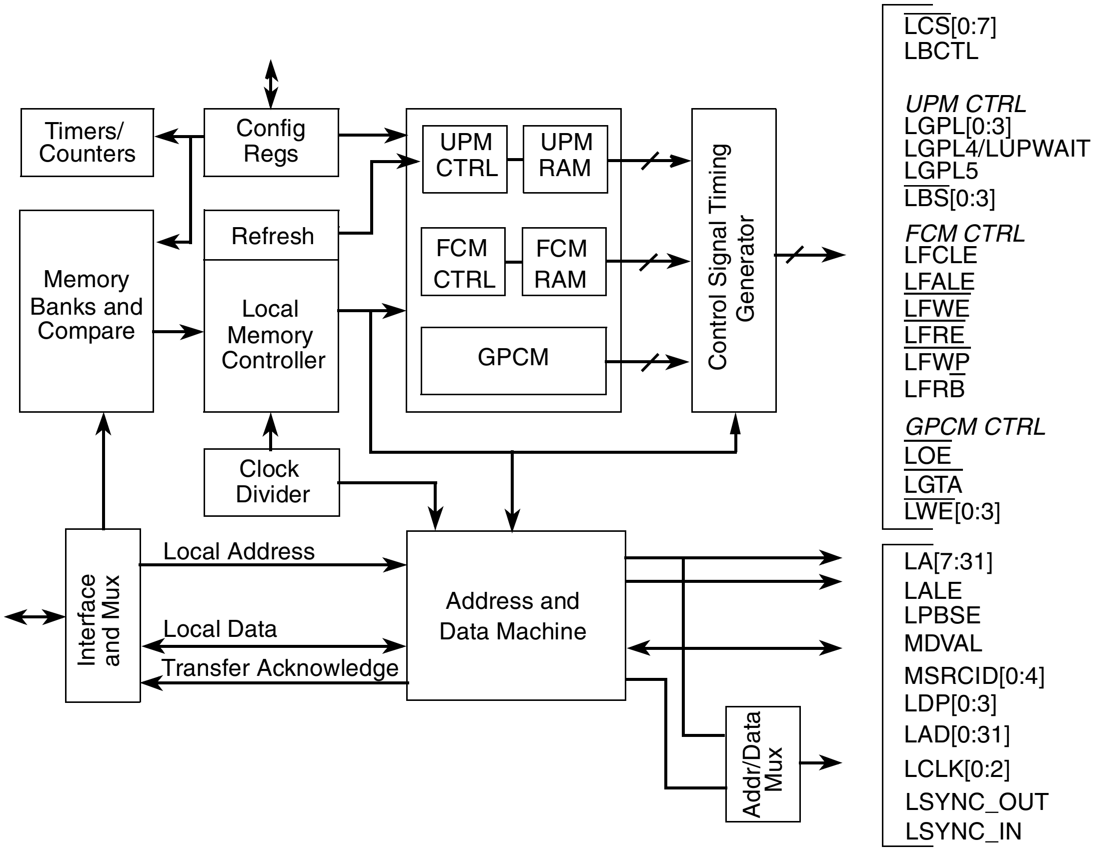


**Figure 13-1. Enhanced Local Bus Controller Block Diagram**

### 13.1.1 Overview

The main component of the eLBC is its memory controller, which provides a seamless interface to many types of memory devices and peripherals. The memory controller is responsible for controlling eight memory banks shared by a GPCM, an FCM, and up to three UPMs. As such, it supports a minimal glue logic interface to SRAM, EPROM, NOR Flash EEPROM, NAND Flash EEPROM, burstable RAM, regular DRAM devices, extended data output DRAM devices, and other peripherals. The external address latch signal (LALE) allows multiplexing of addresses with data signals to reduce the device pin count.The eLBC also includes a number of data checking and protection features such as data parity generation and checking, write protection and a bus monitor to ensure that each bus cycle is terminated within a user-specified period.

### 13.1.2 Features

The eLBC main features are as follows:

- Memory controller with eight memory banks
  - 32-bit address decoding with mask
  - Variable memory block sizes (32 Kbytes to 4 Gbytes)
  - Selection of control signal generation on a per-bank basis
  - Data buffer controls activated on a per-bank basis
  - Automatic segmentation of large transactions into memory accesses optimized for bus width and addressing capability
  - Odd/even parity checking including read-modify-write (RMW) parity for single accesses
  - Write-protection capability
  - Atomic operation
  - Parity byte-select
- General-purpose chip-select machine (GPCM)
  - Compatible with SRAM, EPROM, NOR Flash EEPROM, and peripherals
  - Global (boot) chip-select available at system reset
  - Boot chip-select support for 8-, 16-, and 32-bit devices
  - Minimum three-clock access to external devices
  - Four byte-write-enable signals ($\overline{\text{LWE}}$[0:3])
  - Output enable signal ($\overline{\text{LOE}}$)
  - External access termination signal ($\overline{\text{LGTA}}$)
- NAND Flash control machine (FCM)
  - Compatible with small (512+16 bytes) and large (2048+64 bytes) page parallel NAND Flash EEPROM
  - Global (boot) chip-select available at system reset, with 4-Kbyte boot block buffer for execute-in-place boot loading
  - ECC checking enable/disable feature supported during boot

    - Read-only ECC registers to verify after write operation
    - Boot chip-select support for 8-bit devices
    - Dual 2-Kbyte/eight 512-byte buffers allow simultaneous data transfer during flash reads and programming
    - Interrupt-driven block transfer for reads and writes
    - Programmable command and data transfer sequences of up to eight steps supported
    - Generic command and address registers support proprietary flash interfaces
    - Block write locking to ensure system security and integrity
- Three user-programmable machines (UPMs)
    - Programmable-array-based machine controls external signal timing with a granularity of up to one quarter of an external bus clock period
    - User-specified control-signal patterns run when an internal master requests a single-beat or burst read or write access.
    - UPM refresh timer runs a user-specified control signal pattern to support refresh
    - User-specified control-signal patterns can be initiated by software
    - Each UPM can be defined to support DRAM devices with depths of 64, 128, 256, and 512 Kbytes, and 1, 2, 4, 8, 16, 32, 64, 128, and 256 Mbytes
    - Support for 8-, 16-, and 32-bit devices
    - Page mode support for successive transfers within a burst
    - Internal address multiplexing supporting 64-, 128-, 256-, and 512-Kbyte, and 1-, 2-, 4-, 8-, 16-, 32-, 64-, 128-, and 256-Mbyte page banks
- Optional monitoring of transfers between local bus internal masters and local bus slaves (local bus error reporting on interrupt and status registers)
- Support for phase-locked loop (PLL) with software-configurable bypass for low frequency bus clocks

### 13.1.3 Modes of Operation

The eLBC provides one GPCM, one FCM, and three UPMs for the local bus, with no restriction on how many of the eight banks (chip selects) can be programmed to operate with any given machine. The internal transaction address is limited to 32 bits, so all chip selects must fall within the 4-Gbyte window addressed by the internal transaction address. When a memory transaction is dispatched to the eLBC, the internal transaction address is compared with the address information of each bank (chip select). The corresponding machine assigned to that bank (GPCM, FCM, or UPM) then takes ownership of the external signals that control the access and maintains control until the transaction ends. Thus, with the eLBC in GPCM or FCM, or UPM mode, only one of the eight chip selects is active at any time for the duration of the transaction except in the case of UPM refresh where all UPM machines that are enabled for refresh have concurrent chip select assertion.

#### 13.1.3.1 eLBC Bus Clock and Clock Ratios

The eLBC supports ratios of 4, 8, and 16 between the faster internal (system) clock and slower external bus clock (LCLK[0:2]). This ratio is software programmable through the clock ratio register (LCRR[CLKDIV]). This ratio affects the resolution of signal timing shifts in GPCM and FCM modes and the interpretation of UPM array words in UPM mode. The bus clock is driven identically onto pins, LCLK[0:2], to allow the clock load to be shared equally across a set of signal nets, thereby enhancing the edge rates of the bus clock.

#### 13.1.3.2 Source ID Debug Mode

The eLBC provides the ID of a transaction source on external device pins. When those pins are selected, the 5-bit internal ID of the current transaction source appears on MSRCID[0:4] whenever valid address or data is available on the eLBC external pins. The reserved value of 0x1F, which indicates invalid address or data, appears on the source ID pins at all other times. The combination of a valid source ID (any value except 0x1F) and the value of external address latch enable (LALE) and data valid (MDVAL) facilitate capturing useful debug data as follows:

- If a valid source ID is detected on MSRCID[0:4] and LALE is asserted, a valid full 32-bit address may be latched from LAD[0:26] and combined with LA[27:31].
- If a valid source ID is detected on MSRCID[0:4] and MDVAL is asserted, valid data may be latched from LAD.

The MSRCID[0:4] and MDVAL signals are multiplexed with other functions sharing the same external pins. Refer to  to learn how to enable the MSRCID/MDVAL pins.

## 13.2 External Signal Descriptions

Table 13-1 contains a list of external signals related to the eLBC and summarizes their function. Note that during assertion of $\overline{\text{HRESET}}$, the PLL is initially unlocked, so the LCLK and LSYNC_OUT values are likely to be unstable/jittery for several microseconds; after the PLL locks, stable clock signals are driven on these signals.

**Table 13-1. Signal Properties—Summary**

| Name | Alternate Function(s) | Mode | Descriptions | No. of Signals | I/O |
|---|---|---|---|---|---|
| LALE | — | — | External address latch enable | 1 | O |
| $\overline{\text{LCS}}$[0] | — | — | Chip select 0 | 1 | O |
| $\overline{\text{LCS}}$[1:7] | — | — | Chip selects [1–7] | 7 | O |
| $\overline{\text{LWE0}}$/<br>$\overline{\text{LFWE}}$/<br>$\overline{\text{LBS0}}$ | $\overline{\text{LWE}}$ | GPCM | Write enable 0 | 1 | O |
| | $\overline{\text{LFWE}}$ | FCM | Write enable | 1 | |
| | $\overline{\text{LBS}}$ | UPM | Byte (lane) select 0 | 1 | |
| $\overline{\text{LWE}}$[1:3]/<br>$\overline{\text{LBS}}$[1:3] | $\overline{\text{LWE}}$ | GPCM | Write enable 1–3 | 3 | O |
| | $\overline{\text{LBS}}$ | UPM | Byte (lane) select 1–3 | 3 | |

**Table 13-1. Signal Properties—Summary (continued)**

| Name | Alternate Function(s) | Mode | Descriptions | No. of Signals | I/O |
|---|---|---|---|---|---|
| LGPL0/<br>LFCLE | LGPL0 | UPM | General purpose line 0 | 1 | O |
| | LFCLE | FCM | Flash command latch enable | 1 | |
| LGPL1/<br>LFALE | LGPL1 | UPM | General purpose line 1 | 1 | O |
| | LFALE | FCM | Flash address latch enable | 1 | |
| $\overline{\text{LOE}}$/<br>LGPL2/<br>$\overline{\text{LFRE}}$ | $\overline{\text{LOE}}$ | GPCM | Output enable | 1 | O |
| | $\overline{\text{LFRE}}$ | FCM | Flash read enable | 1 | |
| | LGPL2 | UPM | General purpose line 2 | 1 | |
| LGPL3/<br>$\overline{\text{LFWP}}$ | LGPL3 | UPM | General purpose line 3 | 1 | O |
| | $\overline{\text{LFWP}}$ | FCM | Flash write protect | 1 | |
| $\overline{\text{LGTA}}$/<br>LFR$\overline{\text{B}}$/<br>LGPL4/<br>LUPWAIT/<br>LPBSE | $\overline{\text{LGTA}}$ | GPCM | Transaction termination | 1 | I |
| | LFR$\overline{\text{B}}$ | FCM | Flash ready/$\overline{\text{busy}}$, open-drain shared pin | 1 | I |
| | LGPL4 | UPM | General purpose line 4 | 1 | O |
| | LUPWAIT | UPM | External device wait | 1 | I |
| | LPBSE | — | Local bus parity byte select | 1 | O |
| LGPL5 | — | UPM | General purpose line 5 | 1 | O |
| LBCTL | — | — | Data buffer control | 1 | O |
| LA[7:31] | — | — | Non-multiplexed address bus | 25 | O |
| LAD[0:31] | — | — | Multiplexed address/data bus | 32 | I/O |
| LDP[0:3] | — | — | Local bus data parity | 4 | I/O |
| LCLK[0:2] | — | — | Local bus clocks | 3 | O |
| LSYNC_IN | — | — | PLL synchronize input | 1 | I |
| LSYNC_OUT | — | — | PLL synchronize output | 1 | O |
| MDVAL | — | eLBC debug | Local bus data valid | 1 | O |
| MSRCID[0:4] | — | eLBC debug | Local bus source ID | 5 | O |

Table 13-2 shows the detailed external signal descriptions for the eLBC.

**Table 13-2. Enhanced Local Bus Controller Detailed Signal Descriptions**

<table>
<tr><th>Signal</th><th>I/O</th><th colspan="2">Description</th></tr>
<tr><td rowspan="2">LALE</td><td rowspan="2">O</td><td colspan="2">External address latch enable. The local bus memory controller provides control for an external address latch, which allows address and data to be multiplexed on the device pins.</td></tr>
<tr><td><b>State Meaning</b></td><td>Asserted/Negated—LALE is asserted with the address at the beginning of each memory controller transaction. The number of cycles for which it is asserted is governed by the ORn[EAD] and LCRR[EADC] fields. Note that no other control signals are asserted during the assertion of LALE.</td></tr>
</table>

Table 13-2. Enhanced Local Bus Controller Detailed Signal Descriptions (continued)

| Signal | I/O | Description |
|---|---|---|
| $\overline{\text{LCS}}$[0:7] | O | Chip selects. Eight chip selects are provided that are mutually exclusive. |
| | | **State Meaning** Asserted/Negated—Used to enable specific memory devices or peripherals connected to the eLBC. $\overline{\text{LCS}}$[0:7] are provided on a per-bank basis with $\overline{\text{LCS0}}$ corresponding to the chip select for memory bank 0, which has the memory type and attributes defined by BR0 and OR0. |
| $\overline{\text{LWE0}}$/<br>$\overline{\text{LFWE}}$/<br>$\overline{\text{LBS0}}$,<br>$\overline{\text{LWE}}$[1:3]/<br>LBS[1:3] | O | GPCM write enable 0/FCM write enable/UPM byte select 0. These signals select or validate each byte lane of the data bus. For banks with port sizes of 32 bits (as set by BR*n*[PS]), all four signals are defined. For a 16-bit port size, only bits 0–1 are defined; and for an 8-bit port size, bit 0 is the only defined signal. The least-significant address bits of each access also determine which byte lanes are considered valid for a given data transfer. |
| | | **State Meaning** Asserted/Negated—For GPCM operation, $\overline{\text{LWE}}$[0:3] assert for each byte lane enabled for writing.<br>$\overline{\text{LFWE}}$ enables command, address, and data writes to NAND Flash EEPROMs controlled by FCM.<br>$\overline{\text{LBS}}$[0:3] are programmable byte-select signals in UPM mode. See Section 13.4.4.4, "RAM Array," for programming details about $\overline{\text{LBS}}$[0:3]. |
| | | **Timing** Assertion/Negation—See Section 13.4.2, "General-Purpose Chip-Select Machine (GPCM)," for details regarding the timing of $\overline{\text{LWE}}$[0:3]. |
| LGPL0/<br>LFCLE | O | General purpose line 0/FCM command latch enable. |
| | | **State Meaning** Asserted/Negated—In UPM mode, LGPL0 is one of six general purpose signals; it is driven with a value programmed into the UPM array.<br>In FCM mode, LFCLE enables command cycles to NAND Flash EEPROMs. |
| LGPL1/<br>LFALE | O | General-purpose line 1/FCM address latch enable. |
| | | **State Meaning** Asserted/Negated—In UPM mode, LGPL1 is one of six general purpose signals; it is driven with a value programmed into the UPM array.<br>In FCM mode, LFALE enables address cycles to NAND Flash EPROMs. |
| $\overline{\text{LOE}}$/LGPL2/<br>$\overline{\text{LFRE}}$ | O | GPCM output enable/General-purpose line 2/FCM read enable. |
| | | **State Meaning** Asserted/Negated—Controls the output buffer of memory when accessing memory/devices in GPCM mode.<br>In UPM mode, LGPL2 is one of six general purpose signals; it is driven with a value programmed into the UPM array.<br>$\overline{\text{LFRE}}$ enables data read cycles from NAND Flash EEPROMs controlled by FCM. |
| LGPL3/<br>$\overline{\text{LFWP}}$ | O | General-purpose line 3/FCM write protect. |
| | | **State Meaning** Asserted/Negated—In UPM mode, LGPL3 is one of six general purpose signals; it is driven with a value programmed into the UPM array.<br>In FCM mode $\overline{\text{LFWP}}$ protects NAND Flash EEPROMs from accidental erasure and programming when $\overline{\text{LFWP}}$ is asserted low—see Section 13.3.1.17, "Flash Mode Register (FMR)," for programming of FCM operations to control $\overline{\text{LFWP}}$. |

Table 13-2. Enhanced Local Bus Controller Detailed Signal Descriptions (continued)

| Signal | I/O | Description | |
|---|---|---|---|
| $\overline{\text{LGTA}}$/LGPL4/ LFR$\overline{\text{B}}$/ LUPWAIT/ LPBSE | I/O | GPCM transfer acknowledge/General-purpose line 4/FCM Flash ready-busy/UPM wait/parity byte select. | |
| | | **State Meaning** | Asserted/Negated—Input in GPCM or FCM modes used for transaction termination. It may also be configured as one of six general-purpose output signals when in UPM mode or as an input to force the UPM controller to wait for the memory/device. FCM uses LFR$\overline{\text{B}}$ to stall during long-latency read and programming operations, continuing once LFR$\overline{\text{B}}$ returns high.<br>When configured as LPBSE, it disables any use in GPCM, FCM, or UPM modes. Because systems that use read-modify-write parity require an additional memory device, they must generate a byte-select like a normal data device. ANDing LBS[0:3] through external logic to achieve the logical function of this byte-select can affect memory access timing. The LBC provides this optional byte-select signal connection to RMW-parity devices. |
| LGPL5 | O | General-purpose line 5 | |
| | | **State Meaning** | Asserted/Negated—One of six general purpose signals when in UPM mode, and drives a value programmed in the UPM array. |
| LBCTL | O | Data buffer control. The memory controller activates LBCTL for the local bus when a GPCM-, UPM-, or FCM-controlled bank is accessed. Buffer control is disabled by setting OR*n*[BCTLD]. | |
| | | **State Meaning** | Asserted/Negated—The LBCTL pin normally functions as a write/$\overline{\text{read}}$ control for a bus transceiver connected to the LAD lines. Note that an external data buffer must not drive the LAD lines in conflict with the eLBC when LBCTL is high, because LBCTL remains high after reset and during address phases. |
| LA[7:31] | O | Nonmultiplexed address bus. All bits driven on LA[7:31] are defined for 8-bit port sizes. For 32-bit port sizes, LA[30:31] are don't cares; for 16-bit port sizes LA[31] is a don't care. | |
| | | **State Meaning** | Asserted/Negated—LA is the address bus used to transmit addresses to external RAM devices. Refer to Section 13.5, "Initialization/Application Information," for address signal multiplexing. |
| LAD[0:31] | I/O | Multiplexed address/data bus. For configuration of a port size in BR*n*[PS] as 32 bits, all of LAD[0:31] must be connected to the external RAM data bus, with LAD[0:7] occupying the most significant byte lane (at address offset 0). For a port size of 16 bits, LAD[0:7] connect to the most-significant byte lane (at address offset 0), while LAD[8:15] connect to the least-significant byte lane (at address offset 1); LAD[16:31] are unused for 16-bit port sizes. For a port size of 8 bits, only LAD[0:7] are connected to the external RAM. | |
| | | **State Meaning** | Asserted/Negated—LAD is the shared 32-bit address/data bus through which external RAM devices transfer data and receive addresses. |
| | | **Timing** | Assertion/Negation—During assertion of LALE, LAD are driven with the RAM address for the access to follow. External logic should propagate the address on LAD while LALE is asserted, and latch the address upon negation of LALE. After LALE is negated, LAD are either driven by write data or are made high-impedance by the eLBC in order to sample read data driven by an external device. Following the last data transfer of a write access, LAD are again taken into a high-impedance state. |

Table 13-2. Enhanced Local Bus Controller Detailed Signal Descriptions (continued)

| Signal | I/O | Description | |
|---|---|---|---|
| LDP[0:3] | I/O | Local bus data parity. Drives and receives the data parity corresponding with the data phases on LAD for GPCM and UPM controlled banks. | |
| | | **State Meaning** | Asserted/Negated—During write accesses, a parity bit is generated for each 8 bits of LAD[0:31], such that LDP0 is even/odd parity for LAD[0:7], while LDP[3] is even/odd parity for LAD[24:31]. Unused byte lanes for port sizes less than 32 bits have undefined parity. |
| | | **Timing** | Assertion/Negation—Drive and receive the data parity corresponding with the data phases on LAD. For read accesses, the parity bits for each byte lane are sampled on LDP[0:3] with the same timing that read data is sampled on LAD. LDP[0:3] change impedance in concert with LAD. |
| LCLK[0:2] | O | Local bus clocks | |
| | | **State Meaning** | Asserted/Negated—LCLK[0:2] drive an identical bus clock signal for distributed loads. If the eLBC PLL is enabled (see LCRR[PBYP], Figure 13-19), the bus clock phase is shifted earlier than transitions on other eLBC signals (such as LAD*n* and $\overline{\text{LCS}\mathit{n}}$) by a time delay matching the delay of the PLL timing loop set up between LSYNC_OUT and LSYNC_IN. |
| LSYNC_OUT | O | PLL synchronization out. | |
| | | **State Meaning** | Asserted/Negated—A replica of the bus clock, appearing on LSYNC_OUT, should be propagated through a passive timing loop and returned to LSYNC_IN for achieving correct PLL lock. |
| | | **Timing** | Assertion/Negation—The time delay of the timing loop should be such that it compensates for the round-trip flight time of LCLK[0:2] and clocked drivers in the system. No load other than a timing loop should be placed on LSYNC_OUT. |
| LSYNC_IN | I | PLL synchronization in. | |
| | | **State Meaning** | Asserted/Negated—See description of LSYNC_OUT. |
| MDVAL | O | Local bus data valid (eLBC debug mode only) | |
| | | **State Meaning** | Asserted/Negated—For a read, MDVAL asserts for one bus cycle in the cycle immediately preceding the sampling of read data on LAD. For a write, MDVAL asserts for one bus cycle during the final cycle for which the current write data on LAD is valid. During burst transfers, MDVAL asserts for each data beat. |
| | | **Timing** | Assertion/Negation—Valid only while the eLBC is in system debug mode. In debug mode, MDVAL asserts when the eLBC generates a data transfer acknowledge. |
| MSRCID[0:4] | O | Local bus source ID (eLBC debug mode only). In debug mode, all MSRCID[0:4] pins are driven high unless MSRCID[0:4] is driving a debug source ID for identifying the internal system device controlling the eLBC. | |
| | | **State Meaning** | Asserted/Negated—Remain high until the last bus cycle of the assertion of LALE, in which case the source ID of the address is indicated, or until MDVAL is asserted, in which case the source ID relating to the data transfer is indicated. In case of address debug, MSRCID[0:4] is valid only when the address on LAD consists of all physical address bits—with optional padding—for reconstructing the system address presented to the eLBC. |

## 13.3 Memory Map/Register Definition

Table 13-3 shows the memory mapped registers of the eLBC. Undefined 4-byte address spaces within offset 0x000–0xFFF are reserved.

Table 13-3. Enhanced Local Bus Controller Registers

| Enhanced Local Bus Controller—Block Base Address 0x0_5000 | | | | |
|---|---|---|---|---|
| **Offset** | **Register** | **Access** | **Reset** | **Section/Page** |
| 0x000 | BR0—Base register 0 | R/W | 0x0000_*nnnn* | 13.3.1.1/13-11 |
| 0x008 | BR1—Base register 1 | R/W | 0x0000_0000 | 13.3.1.1/13-11 |
| 0x010 | BR2—Base register 2 | R/W | 0x0000_0000 | 13.3.1.1/13-11 |
| 0x018 | BR3—Base register 3 | R/W | 0x0000_0000 | 13.3.1.1/13-11 |
| 0x020 | BR4—Base register 4 | R/W | 0x0000_0000 | 13.3.1.1/13-11 |
| 0x028 | BR5—Base register 5 | R/W | 0x0000_0000 | 13.3.1.1/13-11 |
| 0x030 | BR6—Base register 6 | R/W | 0x0000_0000 | 13.3.1.1/13-11 |
| 0x038 | BR7—Base register 7 | R/W | 0x0000_0000 | 13.3.1.1/13-11 |
| 0x004 | OR0—Options register 0 | R/W | 0x0000_0FF7 | 13.3.1.2/13-12 |
| 0x00C | OR1—Options register 1 | R/W | 0x0000_0000 | 13.3.1.2/13-12 |
| 0x014 | OR2—Options register 2 | R/W | 0x0000_0000 | 13.3.1.2/13-12 |
| 0x01C | OR3—Options register 3 | R/W | 0x0000_0000 | 13.3.1.2/13-12 |
| 0x024 | OR4—Options register 4 | R/W | 0x0000_0000 | 13.3.1.2/13-12 |
| 0x02C | OR5—Options register 5 | R/W | 0x0000_0000 | 13.3.1.2/13-12 |
| 0x034 | OR6—Options register 6 | R/W | 0x0000_0000 | 13.3.1.2/13-12 |
| 0x03C | OR7—Options register 7 | R/W | 0x0000_0000 | 13.3.1.2/13-12 |
| 0x040–<br>0x064 | Reserved | — | — | — |
| 0x068 | MAR—UPM address register | R/W | 0x0000_0000 | 13.3.1.3/13-20 |
| 0x06C | Reserved | — | — | — |
| 0x070 | MAMR—UPMA mode register | R/W | 0x0000_0000 | 13.3.1.4/13-21 |
| 0x074 | MBMR—UPMB mode register | R/W | 0x0000_0000 | 13.3.1.4/13-21 |
| 0x078 | MCMR—UPMC mode register | R/W | 0x0000_0000 | 13.3.1.4/13-21 |
| 0x07C–<br>0x080 | Reserved | — | — | — |
| 0x084 | MRTPR—Memory refresh timer prescaler register | R/W | 0x0000_0000 | 13.3.1.5/13-23 |
| 0x088 | MDR—UPM/FCM data register | R/W | 0x0000_0000 | 13.3.1.6/13-23 |
| 0x08C | Reserved | — | — | — |

**Table 13-3. Enhanced Local Bus Controller Registers (continued)**

| Enhanced Local Bus Controller—Block Base Address 0x0_5000 | | | | |
|---|---|---|---|---|
| **Offset** | **Register** | **Access** | **Reset** | **Section/Page** |
| 0x090 | LSOR—Special operation initiation register | R/W | 0x0000_0000 | 13.3.1.7/13-24 |
| 0x094–<br>0x09C | Reserved | — | — | — |
| 0x0A0 | LURT—UPM refresh timer | R/W | 0x0000_0000 | 13.3.1.4/13-21 |
| 0x0A4–<br>0x0AC | Reserved | — | — | — |
| 0x0B0 | LTESR—Transfer error status register | w1c | 0x0000_0000 | 13.3.1.9/13-26 |
| 0x0B4 | LTEDR—Transfer error disable register | R/W | 0x0000_0000 | 13.3.1.10/13-28 |
| 0x0B8 | LTEIR—Transfer error interrupt register | R/W | 0x0000_0000 | 13.3.1.11/13-29 |
| 0x0BC | LTEATR—Transfer error attributes register | R/W | 0x0000_0000 | 13.3.1.12/13-30 |
| 0x0C0 | LTEAR—Transfer error address register | R/W | 0x0000_0000 | 13.3.1.13/13-31 |
| 0x0C4 | LTECCR—Transfer error ECC register | w1c | 0x0000_0000 | 13.3.1.14/13-31 |
| 0x0C8–<br>0x0CC | Reserved | — | — | — |
| 0x0D0 | LBCR—Configuration register | R/W | | 13.3.1.15/13-32 |
| 0x0D4 | LCRR—Clock ratio register | R/W | 0x8000_000*n* | 13.3.1.16/13-34 |
| 0x0D8–<br>0x0DC | Reserved | — | — | — |
| 0x0E0 | FMR—Flash mode register | R/W | 0x0000_0*n*00 | 13.3.1.17/13-35 |
| 0x0E4 | FIR—Flash instruction register | R/W | 0x0000_0000 | 13.3.1.18/13-37 |
| 0x0E8 | FCR—Flash command register | R/W | 0x0000_0000 | 13.3.1.19/13-38 |
| 0x0EC | FBAR—Flash block address register | R/W | 0x0000_0000 | 13.3.1.20/13-39 |
| 0x0F0 | FPAR—Flash page address register | R/W | 0x0000_0000 | 13.3.1.21/13-39 |
| 0x0F4 | FBCR—Flash byte count register | R/W | 0x0000_0000 | 13.3.1.22/13-41 |
| 0x0F8–<br>0x0FC | Reserved | — | — | — |
| 0x100 | FECC0—Flash ECC block 0 register | R | 0x0000_0000 | 13.3.1.23/13-41 |
| 0x104 | FECC1—Flash ECC block 1 register | R | 0x0000_0000 | 13.3.1.23/13-41 |
| 0x108 | FECC2—Flash ECC block 2 register | R | 0x0000_0000 | 13.3.1.23/13-41 |
| 0x10C | FECC3—Flash ECC block 3 register | R | 0x0000_0000 | 13.3.1.23/13-41 |

## 13.3.1 Register Descriptions

This section provides a detailed description of the eLBC configuration, status, and control registers with detailed bit and field descriptions.

Address offsets in the eLBC address range that are not defined in Table 13-3 should not be accessed for reading or writing. Similarly, only zero should be written to reserved bits of defined registers, as writing ones can have unpredictable results in some cases.

Bits designated as write-one-to-clear are cleared only by writing ones to them. Writing zeros to them has no effect.

### 13.3.1.1 Base Registers (BR0–BR7)

The base registers (BR*n*), shown in Figure 13-2, contain the base address and address types for each memory bank. The memory controller uses this information to compare the address bus value with the current address accessed. Each register (bank) includes a memory attribute and selects the machine for memory operation handling. Note that after system reset, BR0[V] is set, BR1[V]–BR7[V] are cleared, and the value of BR0[PS] reflects the initial port size configured by the boot ROM location power-on configuration settings.

Offset BR0: 0x0_5000
BR1: 0x0_5008
BR2: 0x0_5010
BR3: 0x0_5018
BR4: 0x0_5020
BR5: 0x0_5028
BR6: 0x0_5030
BR7: 0x0_5038

Access: Read/Write

| | 0–16 | 17–18 | 19–20 | 21–22 | 23 | 24–26 | 27 | 28–29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | BA | — | PS | DECC | WP | MSEL | — | ATOM | — | V |
| Reset | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 0 | P S | n n | 0 | 0 0 M | 0 | 0 0 | 1 | V[1] |

[1] BR0 has its valid bit (V) set at reset. Thus bank 0 is valid with the port size (PS) configured from cfg_rom_loc[0:3] at power-on reset. M = 0 for MSEL of GPCM, 1 for MSEL of FCM at boot.The reset value for DECC is determined by cfg_rom_loc. All other base registers have all bits cleared to zero during reset.

**Figure 13-2. Base Registers (BR*n*)**

Table 13-4 describes BR*n* fields.

**Table 13-4. BR*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–16 | BA | Base address. The upper 17 bits of each base register are compared to the address on the address bus to determine if the bus master is accessing a memory bank controlled by the memory controller. Used with the address mask bits OR*n*[AM]. |
| 17–18 | — | Reserved |
| 19–20 | PS | Port size. Specifies the port size of this memory region. For BR0, PS is configured from the power-on reset configuration setting. For all other banks the value is reset to 00 (port size not defined).<br>00 Reserved<br>01 8-bit (supported for GPCM, UPM, FCM)<br>10 16-bit (supported for GPCM, UPM)<br>11 32-bit (not supported for FCM) |

**Table 13-4. BR*n* Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 21–22 | DECC | Specifies the method for data error checking.<br>00 Data error checking disabled, but normal parity generation for GPCM and UPM. No ECC generation for FCM.<br>01 Normal parity generation and checking for GPCM and UPM. ECC checking is enabled, but ECC generation is disabled, for FCM on full-page transfers.<br>10 Read-modify-write parity generation and normal parity checking for GPCM and UPM. ECC checking and generation are enabled for FCM on full-page transfers.<br>11 Reserved |
| 23 | WP | Write protect.<br>0 Read and write accesses are allowed.<br>1 Only read accesses are allowed. The memory controller does not assert $\overline{\text{LCS}n}$ on write cycles to this memory bank. LTESR[WP] is set (if WP is set) if a write to this memory bank is attempted, and a local bus error interrupt is generated (if enabled), terminating the cycle. |
| 24–26 | MSEL | Machine select. Specifies the machine to use for handling memory operations.<br>000 GPCM (possible reset value)<br>001 FCM (possible reset value)<br>010 Reserved<br>011 Reserved<br>100 UPMA<br>101 UPMB<br>110 UPMC<br>111 Reserved |
| 27 | — | Reserved |
| 28–29 | ATOM | Atomic operation. Writes (reads) to the address space handled by the memory controller bank reserve the selected memory bank for the exclusive use of the accessing device. The reservation is released when the device performs a read (write) operation to this memory controller bank. If a subsequent read (write) request to this memory controller bank is not detected within 256 bus clock cycles of the last write (read), the reservation is released and an atomic error is reported (if enabled).<br>00 The address space controlled by this bank is not used for atomic operations.<br>01 Read-after-write-atomic (RAWA).<br>10 Write-after-read-atomic (WARA).<br>11 Reserved |
| 30 | — | Reserved |
| 31 | V | Valid bit. Indicates that the contents of the BR*n* and OR*n* pair are valid. $\overline{\text{LCS}n}$ does not assert unless V is set (an access to a region that has no valid bit set may cause a bus time-out). After a system reset, only BR0[V] is set.<br>0 This bank is invalid.<br>1 This bank is valid. |

### 13.3.1.2 Option Registers (OR0–OR7)

The OR*n* registers define the sizes of memory banks and access attributes. The OR*n* attribute bits support the following three modes of operation as defined by BR*n*[MSEL]:

- GPCM mode
- FCM mode
- UPM mode

The OR*n* registers are interpreted differently depending on which of the three machine types is selected for that bank. Because bank 0 can be used to boot, the reset value of OR0 may be different depending on power-on configuration options. Table 13-5 shows the reset values for OR0.

Table 13-5. Reset value of OR0 Register

| Boot Source | OR0 Reset Value |
|---|---|
| FCM (small page NAND Flash) | 0000_03AE |
| FCM (large page NAND Flash) | 0000_07AE |
| GPCM | 0000_0FF7 |
| eLBC not used as a boot source | 0000_0F07 |

#### 13.3.1.2.1 Address Mask

The address mask field of the option registers (OR*n*[AM]) masks up to 17 corresponding BR*n*[BA] fields. The 15 LSBs of the 32-bit internal transaction address do not participate in bank address matching in selecting a bank for access. Masking address bits independently allows external devices of different size address ranges to be used. Address mask bits can be set or cleared in any order in the field, allowing a resource to reside in more than one area of the address map. Table 13-6 shows memory bank sizes from 32 Kbytes to 4 Gbytes.

Table 13-6. Memory Bank Sizes in Relation to Address Mask

| AM | Memory Bank Size |
|---|---|
| 0000_0000_0000_0000_0 | 4 Gbytes |
| 1000_0000_0000_0000_0 | 2 Gbytes |
| 1100_0000_0000_0000_0 | 1 Gbyte |
| 1110_0000_0000_0000_0 | 512 Mbytes |
| 1111_0000_0000_0000_0 | 256 Mbytes |
| 1111_1000_0000_0000_0 | 128 Mbytes |
| 1111_1100_0000_0000_0 | 64 Mbytes |
| 1111_1110_0000_0000_0 | 32 Mbytes |
| 1111_1111_0000_0000_0 | 16 Mbytes |
| 1111_1111_1000_0000_0 | 8 Mbytes |
| 1111_1111_1100_0000_0 | 4 Mbytes |
| 1111_1111_1110_0000_0 | 2 Mbytes |
| 1111_1111_1111_0000_0 | 1 Mbyte |
| 1111_1111_1111_1000_0 | 512 Kbytes |
| 1111_1111_1111_1100_0 | 256 Kbytes |
| 1111_1111_1111_1110_0 | 128 Kbytes |

**Table 13-6. Memory Bank Sizes in Relation to Address Mask (continued)**

| AM | Memory Bank Size |
|---|---|
| 1111_1111_1111_1111_0 | 64 Kbytes |
| 1111_1111_1111_1111_1 | 32 Kbytes |

#### 13.3.1.2.2 Option Registers (OR*n*)—GPCM Mode

Figure 13-3 shows the bit fields for OR*n* when the corresponding BR*n*[MSEL] selects the GPCM machine.

Offset OR0: 0x0_5004
OR1: 0x0_500c
OR2: 0x0_5014
OR3: 0x0_501c
OR4: 0x0_5024
OR5: 0x0_502c
OR6: 0x0_5034
OR7: 0x0_503c

Access: Read/Write

| | 0 … 15 |
|---|---|
| R<br>W | AM |
| Reset | All zeros |

| | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | AM | — | | BCTLD | CSNT | ACS | | XACS | SCY | | SETA | TRLX | EHTR | EAD |
| Reset | All zeros[1] | | | | | | | | | | | | | |

[1] Refer to Table 13-5 for the OR0 reset value. All other option registers have all bits cleared.

**Figure 13-3. Option Registers (OR*n*) in GPCM Mode**

Table 13-7 describes OR*n* fields for GPCM mode.

**Table 13-7. OR*n*—GPCM Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–16 | AM | GPCM address mask. Masks corresponding BR*n* bits. Masking address bits independently allows external devices of different size address ranges to be used. Address mask bits can be set or cleared in any order in the field, allowing a resource to reside in more than one area of the address map.<br>0 Corresponding address bits are masked and therefore don't care for address checking.<br>1 Corresponding address bits are used in the comparison between base and transaction addresses. |
| 17–18 | — | Reserved |
| 19 | BCTLD | Buffer control disable. Disables assertion of LBCTL during access to the current memory bank.<br>0 LBCTL is asserted upon access to the current memory bank.<br>1 LBCTL is not asserted upon access to the current memory bank. |
| 20 | CSNT | Chip select negation time. Determines when $\overline{\mathrm{LCS}n}$ and $\overline{\mathrm{LWE}}$ are negated during an external memory write access handled by the GPCM, provided that ACS ≠ 00 (when ACS = 00, only $\overline{\mathrm{LWE}}$ is affected by the setting of CSNT). This helps meet address/data hold times for slow memories and peripherals.<br>0 $\overline{\mathrm{LCS}n}$ and $\overline{\mathrm{LWE}}$ are negated normally.<br>1 $\overline{\mathrm{LCS}n}$ and $\overline{\mathrm{LWE}}$ are negated one quarter of a bus clock cycle earlier. |

**Table 13-7. OR*n*—GPCM Field Descriptions (continued)**

<table>
<tr><th>Bits</th><th>Name</th><th>Description</th></tr>
<tr><td>21–22</td><td>ACS</td><td>Address to chip-select setup. Determines the delay of the $\overline{\text{LCS}n}$ assertion relative to the address change when the external memory access is handled by the GPCM. At system reset, OR0[ACS] = 11.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>00</td><td>$\overline{\text{LCS}n}$ is output at the same time as the address lines. Note that this overrides the value of CSNT such that CSNT = 0.</td></tr>
<tr><td>01</td><td>Reserved.</td></tr>
<tr><td>10</td><td>$\overline{\text{LCS}n}$ is output one quarter bus clock cycle after the address lines.</td></tr>
<tr><td>11</td><td>$\overline{\text{LCS}n}$ is output one half bus clock cycle after the address lines.</td></tr>
</table>
</td></tr>
<tr><td>23</td><td>XACS</td><td>Extra address to chip-select setup. Setting this bit increases the delay of the $\overline{\text{LCS}n}$ assertion relative to the address change when the external memory access is handled by the GPCM. After a system reset, OR0[XACS] = 1.<br>0 Address to chip-select setup is determined by ORx[ACS].<br>1 Address to chip-select setup is extended (see Table 13-32 and Table 13-33).</td></tr>
<tr><td>24–27</td><td>SCY</td><td>Cycle length in bus clocks. Determines the number of wait states inserted in the bus cycle, when the GPCM handles the external memory access. Thus it is the main parameter for determining cycle length. The total cycle length depends on other timing attribute settings. After a system reset, OR0[SCY] = 1111.<br>0000 No wait states<br>0001 1 bus clock cycle wait state<br>...<br>1111 15 bus clock cycle wait states</td></tr>
<tr><td>28</td><td>SETA</td><td>External address termination.<br>0 Access is terminated internally by the memory controller unless the external device asserts $\overline{\text{LGTA}}$ earlier to terminate the access.<br>1 Access is terminated externally by asserting the $\overline{\text{LGTA}}$ external pin. (Only $\overline{\text{LGTA}}$ can terminate the access).</td></tr>
<tr><td>29</td><td>TRLX</td><td>Timing relaxed. Modifies the settings of timing parameters for slow memories or peripherals.<br>0 Normal timing is generated by the GPCM.<br>1 Relaxed timing on the following parameters:<br>• Adds an additional cycle between the address and control signals (only if ACS is not equal to 00).<br>• Doubles the number of wait states specified by SCY, providing up to 30 wait states.<br>• Works in conjunction with EHTR to extend hold time on read accesses.<br>• $\overline{\text{LCS}n}$ (only if ACS is not equal to 00) and $\overline{\text{LWE}}$ signals are negated one cycle earlier during writes.</td></tr>
</table>

**Table 13-7. OR*n*—GPCM Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 30 | EHTR | Extended hold time on read accesses. Indicates with TRLX how many cycles are inserted between a read access from the current bank and the next access.<br><br>TRLX / EHTR / Meaning:<br>0 / 0 / The memory controller generates normal timing. No additional cycles are inserted.<br>0 / 1 / 1 idle clock cycle is inserted.<br>1 / 0 / 4 idle clock cycles are inserted.<br>1 / 1 / 8 idle clock cycles are inserted. |
| 31 | EAD | External address latch delay. Allow extra bus clock cycles when using external address latch (LALE).<br>0 No additional bus clock cycles (LALE asserted for one bus clock cycle only)<br>1 Extra bus clock cycles are added (LALE is asserted for the number of bus clock cycles specified by LCRR[EADC]). |

### 13.3.1.2.3 Option Registers (OR*n*)—FCM Mode

Figure 13-4 shows the bit fields for OR*n* when the corresponding BR*n*[MSEL] selects the FCM machine.

Offset OR0: 0x0_5004
OR1: 0x0_500c
OR2: 0x0_5014
OR3: 0x0_501c
OR4: 0x0_5024
OR5: 0x0_502c
OR6: 0x0_5034
OR7: 0x0_503c

Access: Read/Write

| | 0–15 |
|---|---|
| R<br>W | AM |
| Reset | All zeros |

| | 16 | 17–18 | 19 | 20 | 21 | 22 | 23 | 24 | 25–27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | AM | — | BCTLD | — | PGS | CSCT | CST | CHT | SCY | RST | TRLX | EHTR | — |
| Reset | All zeros[1] | | | | | | | | | | | | |

[1] Refer to Table 13-5 for the OR0 reset value. All other option registers have all bits cleared.

**Figure 13-4. Option Registers (OR*n*) in FCM Mode**

Table 13-8 describes OR*n* fields for FCM mode.

**Table 13-8. OR*n*—FCM Field Descriptions**

<table>
<tr><th>Bits</th><th>Name</th><th>Description</th></tr>
<tr><td>0–16</td><td>AM</td><td>FCM address mask. Masks corresponding BR<i>n</i> bits. Masking address bits independently allows external devices of different size address ranges to be used. Address mask bits can be set or cleared in any order in the field, allowing a resource to reside in more than one area of the address map.<br>0 Corresponding address bits are masked.<br>1 Corresponding address bits are used in the comparison between base and transaction addresses.</td></tr>
<tr><td>17–18</td><td>—</td><td>Reserved</td></tr>
<tr><td>19</td><td>BCTLD</td><td>Buffer control disable. Disables assertion of LBCTL during access to the current memory bank.<br>0 LBCTL is asserted upon access to the current memory bank.<br>1 LBCTL is not asserted upon access to the current memory bank.</td></tr>
<tr><td>20</td><td>—</td><td>Reserved</td></tr>
<tr><td>21</td><td>PGS</td><td>NAND Flash EEPROM page size, buffer size, and block size.<br>0 Page size of 512 main area bytes plus 16 spare area bytes (small page devices); FCM RAM buffers are 1 Kbyte each; Flash block size of 16 Kbytes.<br>1 Page size of 2048 main area bytes plus 64 spare area bytes (large page devices); FCM RAM buffers are 4 Kbytes each; Flash block size of 128 Kbytes.</td></tr>
<tr><td>22</td><td>CSCT</td><td>Chip select to command time. Determines how far in advance $\overline{\mathrm{LCS}n}$ is asserted prior to any bus activity during a NAND Flash access handled by the FCM. This helps meet chip-select setup times for slow memories.
<table>
<tr><th>TRLX</th><th>CSCT</th><th>Meaning</th></tr>
<tr><td>0</td><td>0</td><td>The chip-select is asserted 1 clock cycle before any command.</td></tr>
<tr><td>0</td><td>1</td><td>The chip-select is asserted 4 clock cycles before any command.</td></tr>
<tr><td>1</td><td>0</td><td>The chip-select is asserted 2 clock cycles before any command.</td></tr>
<tr><td>1</td><td>1</td><td>The chip-select is asserted 8 clock cycles before any command.</td></tr>
</table></td></tr>
<tr><td>23</td><td>CST</td><td>Command setup time. Determines the delay of $\overline{\mathrm{LFWE}}$ assertion relative to the command, address, or data change when the external memory access is handled by the FCM.
<table>
<tr><th>TRLX</th><th>CST</th><th>Meaning</th></tr>
<tr><td>0</td><td>0</td><td>The write-enable is asserted coincident with any command.</td></tr>
<tr><td>0</td><td>1</td><td>The write-enable is asserted 0.25 clock cycles after any command, address, or data.</td></tr>
<tr><td>1</td><td>0</td><td>The write-enable is asserted 0.5 clock cycles after any command, address, or data.</td></tr>
<tr><td>1</td><td>1</td><td>The write-enable is asserted 1 clock cycle after any command, address, or data.</td></tr>
</table></td></tr>
</table>

Table 13-8. OR*n*—FCM Field Descriptions (continued)

<table>
<tr><th>Bits</th><th>Name</th><th>Description</th></tr>
<tr><td>24</td><td>CHT</td><td>Command hold time. Determines the $\overline{\text{LFWE}}$ negation prior to the command, address, or data change when the external memory access is handled by the FCM.
<table>
<tr><th>TRLX</th><th>CHT</th><th>Meaning</th></tr>
<tr><td>0</td><td>0</td><td>The write-enable is negated 0.5 clock cycles before any command, address, or data change.</td></tr>
<tr><td>0</td><td>1</td><td>The write-enable is negated 1 clock cycle before any command, address, or data change.</td></tr>
<tr><td>1</td><td>0</td><td>The write-enable is negated 1.5 clock cycles before any command, address, or data change.</td></tr>
<tr><td>1</td><td>1</td><td>The write-enable is negated 2 clock cycles before any command, address, or data change.</td></tr>
</table></td></tr>
<tr><td>25–27</td><td>SCY</td><td>Cycle length in bus clocks. Determines:<br>• the number of wait states inserted in command, address, or data transfer bus cycles, when the FCM handles the external memory access. Thus it is the main parameter for determining cycle length. The total cycle length depends on other timing attribute settings.<br>• the delay between command/address writes and data write cycles, or the delay between write cycles and read cycles from NAND Flash EEPROM. A delay of 4×(2+SCY) clock cycles (TRLX = 0) or 8×(2+SCY) clock cycles (TRLX = 1) is inserted between the last write and the first data transfer to/from NAND Flash devices.<br>• the delay between a command write and the first sample point of the RDY/$\overline{\text{BSY}}$ pin (connected to LFR$\overline{\text{B}}$). LFR$\overline{\text{B}}$ is not sampled until 8×(2+SCY) clock cycles (TRLX = 0) or 16×(2+SCY) clock cycles (TRLX = 1) have elapsed following the command.<br>000 No extra wait states<br>001 1 bus clock cycle wait state<br>...<br>111 7 bus clock cycle wait states</td></tr>
<tr><td>28</td><td>RST</td><td>Read setup time. Determines the delay of $\overline{\text{LFRE}}$ assertion relative to sampling of read data when the external memory access is handled by the FCM.
<table>
<tr><th>TRLX</th><th>RST</th><th>Meaning</th></tr>
<tr><td>0</td><td>0</td><td>The read-enable is asserted 0.75 clock cycles prior to any wait states.</td></tr>
<tr><td>0</td><td>1</td><td>The read-enable is asserted 1 clock cycle prior to any wait states.</td></tr>
<tr><td>1</td><td>0</td><td>The read-enable is asserted 0.5 clock cycles prior to any wait states.</td></tr>
<tr><td>1</td><td>1</td><td>The read-enable is asserted 1 clock cycle prior to any wait states.</td></tr>
</table></td></tr>
<tr><td>29</td><td>TRLX</td><td>Timing relaxed. Modifies the settings of timing parameters for slow memories.<br>0 Normal timing is generated by the FCM.<br>1 Relaxed timing on the following parameters:<br>• Doubles the number of clock cycles between $\overline{\text{LCS}n}$ assertion and commands.<br>• Doubles the number of wait states specified by SCY, providing up to 14 wait states.<br>• Works in conjunction with CST and RST to extend command/address/data setup times.<br>• Adds one clock cycle to the command/address/data hold times.<br>• Works in conjunction with CBT to extend the wait time for read/busy status sampling by 16 clock cycles.<br>• Works in conjunction with EHTR to double hold time on read accesses.</td></tr>
</table>

Table 13-8. OR*n*—FCM Field Descriptions (continued)

<table>
<tr><th>Bits</th><th>Name</th><th>Description</th></tr>
<tr><td>30</td><td>EHTR</td><td>Extended hold time on read accesses. Indicates with TRLX how many cycles are inserted between a read access from the current bank and the next access.
<table>
<tr><th>TRLX</th><th>EHTR</th><th>Meaning</th></tr>
<tr><td>0</td><td>0</td><td>1 idle clock cycle is inserted.</td></tr>
<tr><td>0</td><td>1</td><td>2 idle clock cycles are inserted.</td></tr>
<tr><td>1</td><td>0</td><td>4 idle clock cycles are inserted.</td></tr>
<tr><td>1</td><td>1</td><td>8 idle clock cycles are inserted.</td></tr>
</table>
</td></tr>
<tr><td>31</td><td>—</td><td>Reserved</td></tr>
</table>

### 13.3.1.2.4 Option Registers (OR*n*)—UPM Mode

Figure 13-5 shows the bit fields for OR*n* when the corresponding BR*n*[MSEL] selects a UPM machine.

Offset OR0: 0x0_5004
OR1: 0x0_500c
OR2: 0x0_5014
OR3: 0x0_501c
OR4: 0x0_5024
OR5: 0x0_502c
OR6: 0x0_5034
OR7: 0x0_503c

Access: Read/Write

| | 0–15 |
|---|---|
| R<br>W | AM |
| Reset | All zeros |

| | 16 | 17–18 | 19 | 20–22 | 23 | 24–28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|
| R<br>W | AM | — | BCTLD | — | BI | — | TRLX | EHTR | EAD |
| Reset | All zeros[1] | | | | | | | | |

[1] Refer to Table 13-5 for the OR0 reset value. All other option registers have all bits cleared.

Figure 13-5. Option Registers (OR*n*) in UPM Mode

Table 13-9 describes BR*n* fields for UPM mode.

**Table 13-9. OR*n*—UPM Field Descriptions**

<table>
<tr><th>Bits</th><th>Name</th><th>Description</th></tr>
<tr><td>0–16</td><td>AM</td><td>UPM address mask. Masks corresponding BR<i>n</i> bits. Masking address bits independently allows external devices of different size address ranges to be used. Address mask bits can be set or cleared in any order in the field, allowing a resource to reside in more than one area of the address map.<br>0 Corresponding address bits are masked.<br>1 The corresponding address bits are used in the comparison with address pins.</td></tr>
<tr><td>17–18</td><td>—</td><td>Reserved</td></tr>
<tr><td>19</td><td>BCTLD</td><td>Buffer control disable. Disables assertion of LBCTL during access to the current memory bank.<br>0 LBCTL is asserted upon access to the current memory bank.<br>1 LBCTL is not asserted upon access to the current memory bank.</td></tr>
<tr><td>20–22</td><td>—</td><td>Reserved</td></tr>
<tr><td>23</td><td>BI</td><td>Burst inhibit. Indicates if this memory bank supports burst accesses.<br>0 The bank supports burst accesses.<br>1 The bank does not support burst accesses. The selected UPM executes burst accesses as a series of single accesses.</td></tr>
<tr><td>24–28</td><td>—</td><td>Reserved</td></tr>
<tr><td>29</td><td>TRLX</td><td>Timing relaxed. Works in conjunction with EHTR to extend hold time on read accesses.</td></tr>
<tr><td>30</td><td>EHTR</td><td>Extended hold time on read accesses. Indicates with TRLX how many cycles are inserted between a read access from the current bank and the next access.
<table>
<tr><th>TRLX</th><th>EHTR</th><th>Meaning</th></tr>
<tr><td>0</td><td>0</td><td>The memory controller generates normal timing. No additional cycles are inserted.</td></tr>
<tr><td>0</td><td>1</td><td>1 idle clock cycle is inserted.</td></tr>
<tr><td>1</td><td>0</td><td>4 idle clock cycles are inserted.</td></tr>
<tr><td>1</td><td>1</td><td>8 idle clock cycles are inserted.</td></tr>
</table></td></tr>
<tr><td>31</td><td>EAD</td><td>External address latch delay. Allow extra bus clock cycles when using external address latch (LALE).<br>0 No additional bus clock cycles (LALE asserted for one bus clock cycle only)<br>1 Extra bus clock cycles are added (LALE is asserted for the number of bus clock cycles specified by LCRR[EADC]).</td></tr>
</table>

### 13.3.1.3 UPM Memory Address Register (MAR)

Figure 13-6 shows the fields of the UPM memory address register (MAR).

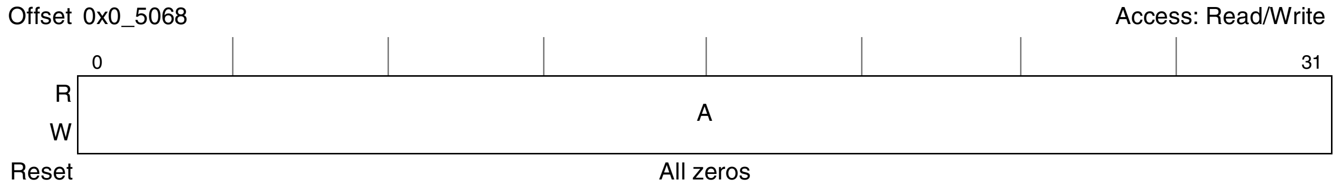


**Figure 13-6. UPM Memory Address Register (MAR)**

Table 13-10 describes the MAR fields.

**Table 13-10. MAR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–31 | A | Address that can be output to the address signals under control of the AMX bits in the UPM RAM word. |

### 13.3.1.4 UPM Mode Registers (M*x*MR)

The UPM machine mode registers (MAMR, MBMR and MCMR), shown in Figure 13-7, contain the configuration for the three UPMs.

Offset MAMR: 0x0_5070
MBMR: 0x0_5074
MCMR: 0x0_5078

Access: Read/Write

| | 0 | 1 | 2 | 3 | 4 | 5 | 7 | 8 | 9 | 10 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | RFEN | OP | | UWPL | AM | | DS | | G0CL | | GPL4 | RLF | |
| Reset | All zeros | | | | | | | | | | | | | |

| | 16 | 17 | 18 | 21 | 22 | 25 | 26 | 31 |
|---|---|---|---|---|---|---|---|---|
| R<br>W | RLF | | WLF | | TLF | | MAD | |
| Reset | All zeros | | | | | | | |

**Figure 13-7. UPM Mode Registers (M*x*MR)**

Table 13-11 describes UPM mode fields.

**Table 13-11. M*x*MR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | — | Reserved |
| 1 | RFEN | Refresh enable. Indicates that the UPM needs refresh services. This bit must be set for UPMA (refresh executor) if refresh services are required on any UPM assigned chip selects. If MAMR[RFEN] = 0, no refresh services can be provided, even if UPMB and/or UPMC have their RFEN bit set.<br>0 Refresh services are not required<br>1 Refresh services are required |
| 2–3 | OP | Command opcode. Determines the command executed by the UPM*n* when a memory access hits a UPM assigned bank.<br>00 Normal operation<br>01 Write to UPM array. On the next memory access that hits a UPM assigned bank, write the contents of the MDR into the RAM location pointed to by MAD. After the access, MAD is automatically incremented.<br>10 Read from UPM array. On the next memory access that hits a UPM assigned bank, read the contents of the RAM location pointed to by MAD into the MDR. After the access, MAD is automatically incremented.<br>11 Run pattern. On the next memory access that hits a UPM assigned bank, run the pattern written in the RAM array. The pattern run starts at the location pointed to by MAD and continues until the LAST bit is set in the RAM word. |
| 4 | UWPL | LUPWAIT polarity active low. Sets the polarity of the LUPWAIT pin when in UPM mode.<br>0 LUPWAIT is active high.<br>1 LUPWAIT is active low. |

**Table 13-11. M*x*MR Field Descriptions (continued)**

<table>
<tr><th>Bits</th><th>Name</th><th>Description</th></tr>
<tr><td>5–7</td><td>AM</td><td>Address multiplex size. Determines how the address of the current memory cycle can be output on the address pins. This field is needed when interfacing with devices requiring row and column addresses multiplexed on the same pins. See Section 13.4.4.4.7, “Address Multiplexing (AMX)” for more information.<br>000 Internal transaction address a[8:23] driven on LAD[16:31]; LAD[0:15] driven low.<br>001 Internal transaction address a[7:22] driven on LAD[16:31]; LAD[0:15] driven low.<br>010 Internal transaction address a[6:21] driven on LAD[16:31]; LAD[0:15] driven low.<br>011 Internal transaction address a[5:20] driven on LAD[16:31]; LAD[0:15] driven low.<br>100 Internal transaction address a[4:19] driven on LAD[16:31]; LAD[0:15] driven low.<br>101 Internal transaction address a[3:18] driven on LAD[16:31]; LAD[0:15] driven low.<br>110 Reserved<br>111 Reserved</td></tr>
<tr><td>8–9</td><td>DS</td><td>Disable timer period. Guarantees a minimum time between accesses to the same memory bank controlled by UPM*n*. The disable timer is turned on by the TODT bit in the RAM array word, and when expired, the UPM*n* allows the machine access to handle a memory pattern to the same bank. Accesses to a different bank by the same UPM*n* is also allowed. To avoid conflicts between successive accesses to different banks, the minimum pattern in the RAM array for a request serviced, should not be shorter than the period established by DS.<br>00 1-bus clock cycle disable period<br>01 2-bus clock cycle disable period<br>10 3-bus clock cycle disable period<br>11 4-bus clock cycle disable period</td></tr>
<tr><td>10–12</td><td>G0CL</td><td>General line 0 control. Determines which logical address line can be output to the LGPL0 pin when the UPM*n* is selected to control the memory access.<br>000 A12<br>001 A11<br>010 A10<br>011 A9<br>100 A8<br>101 A7<br>110 A6<br>111 A5</td></tr>
<tr><td>13</td><td>GPL4</td><td>LGPL4 output line disable. Determines how the LGPL4/LUPWAIT pin is controlled by the corresponding bits in the UPM*n* array. See Table 13-40 on page 13-81.
<table>
<tr><th rowspan="2">Value</th><th rowspan="2">LGPL4/LUPWAIT Pin Function</th><th colspan="2">Interpretation of UPM Word Bits</th></tr>
<tr><th>G4T1/DLT3</th><th>G4T3/WAEN</th></tr>
<tr><td>0</td><td>LGPL4 (output)</td><td>G4T1</td><td>G4T3</td></tr>
<tr><td>1</td><td>LUPWAIT (input)</td><td>DLT3</td><td>WAEN</td></tr>
</table></td></tr>
<tr><td>14–17</td><td>RLF</td><td>Read loop field. Determines the number of times a loop defined in the UPM*n* will be executed for a burst- or single-beat read pattern or when M*x*MR[OP] = 11 (RUN command)<br>0000 16<br>0001 1<br>0010 2<br>0011 3<br>...<br>1110 14<br>1111 15</td></tr>
</table>

Table 13-11. M*x*MR Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 18–21 | WLF | Write loop field. Determines the number of times a loop defined in the UPM*n* will be executed for a burst- or single-beat write pattern.<br>0000  16<br>0001  1<br>0010  2<br>0011  3<br>...<br>1110  14<br>1111  15 |
| 22–25 | TLF | Refresh loop field. Determines the number of times a loop defined in the UPM*n* will be executed for a refresh service pattern.<br>0000  16<br>0001  1<br>0010  2<br>0011  3<br>...<br>1110  14<br>1111  15 |
| 26–31 | MAD | Machine address. RAM address pointer for the command executed. This field is incremented by 1, each time the UPM is accessed and the OP field is set to WRITE or READ. Address range is 64 words per UPM*n*. |

### 13.3.1.5 Memory Refresh Timer Prescaler Register (MRTPR)

The refresh timer prescaler register (MRTPR), shown in Figure 13-8, is used to divide the system clock to provide the UPM refresh timers clock.

Offset 0x0_5084 Access: Read/Write

| | 0–7 | 8–31 |
|---|---|---|
| R<br>W | PTP | — |
| Reset | All zeros | |

Figure 13-8. Memory Refresh Timer Prescaler Register (MRTPR)

Table 13-12 describes MRTPR fields.

Table 13-12. MRTPR Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–7 | PTP | Refresh timers prescaler. Determines the period of the refresh timers input clock. The system clock is divided by PTP except when the value is 00000_0000, which represents the maximum divider of 256. |
| 8–31 | — | Reserved |

### 13.3.1.6 UPM/FCM Data Register (MDR)

The memory data register (MDR), shown in Figure 13-9 and Figure 13-10, contains data written to or read from the RAM array for UPM read or write commands. MDR also contains data written to or read from an external NAND Flash EEPROM for FCM write address, write data, and read status commands. MDR must be set up before issuing a write command to the UPM, or before issuing a FCM operation sequence that uses MDR to source address or data bytes.

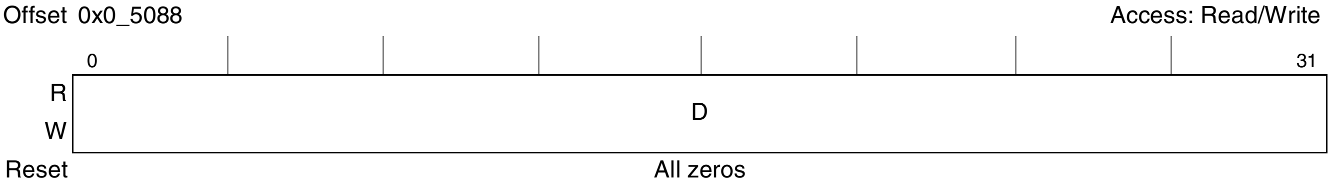


**Figure 13-9. UPM Data Register in UPM Mode (MDR)**

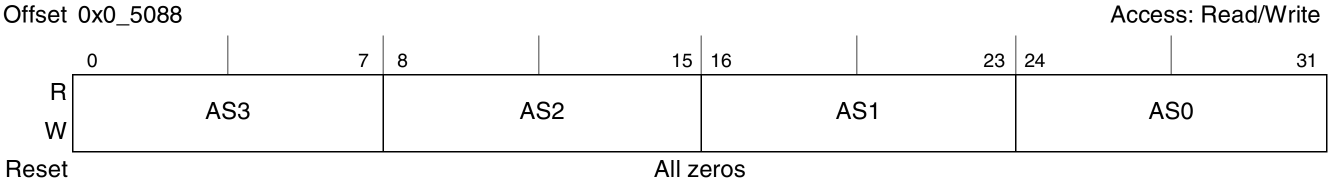


**Figure 13-10. FCM Data Register in FCM Mode (MDR)**

Table 13-13 describes MDR[D].

**Table 13-13. MDR Field Description**

| Bits | Name | Description |
|---|---|---|
| 0–31 | D | In UPM mode, D is the data to be read or written into the RAM array when a write or read command is supplied to the UPM (M*x*MR[OP] = 01 or M*x*MR[OP] = 10). |
| 0–7 | AS3 | In FCM mode, AS3 is the fourth byte of address sent by a custom address write operation, or the fourth byte of data read from a read status operation. |
| 8–15 | AS2 | In FCM mode, AS2 is the third byte of address sent by a custom address write operation, or the third byte of data read from a read status operation. |
| 16–23 | AS1 | In FCM mode, AS1 is the second byte of address sent by a custom address write operation, or the second byte of data read from a read status operation. |
| 24–31 | AS0 | In FCM mode, AS0 is the first byte of address sent by a custom address write operation, or the first byte of data read from a read status operation. |

### 13.3.1.7 Special Operation Initiation Register (LSOR)

The special operation initiation register (LSOR), shown in Figure 13-11, is used by software to trigger a special operation on the indicated bank. Writing to LSOR activates a special operation on bank LSOR[BANK] provided that the bank is valid and controlled by a memory controller whose mode OP field is set to a value other than ‘normal operation.’ If eLBC is currently busy with a memory transaction, writing LSOR completes immediately, but the special operation request is queued until eLBC can service it. To avoid race conditions between software and a busy eLBC, registers that affect currently running special operation and LSOR must not be re-written before a pending special operation has been completed. The UPM and FCM have different indications of when such special operations are completed. The behavior of eLBC is unpredictable if special operation modes are altered between LSOR being written and the relevant memory controller completing that access.

UPM special operation modes are set in registers M*x*MR[OP], see Section 13.3.1.4, “UPM Mode Registers (MxMR).” FCM special operation modes are set in FMR[OP], see Section 13.3.1.17, “Flash Mode Register (FMR).” Writing LSOR has the same effect as setting a special controller mode and performing a dummy access to a bank associated with the controller in question, but use of LSOR avoids changing settings for the address space occupied by the bank. More details of special operation sequences appear in Section 13.4.4.2.1, “UPM Programming Example (Two Sequential Writes to the RAM Array).”

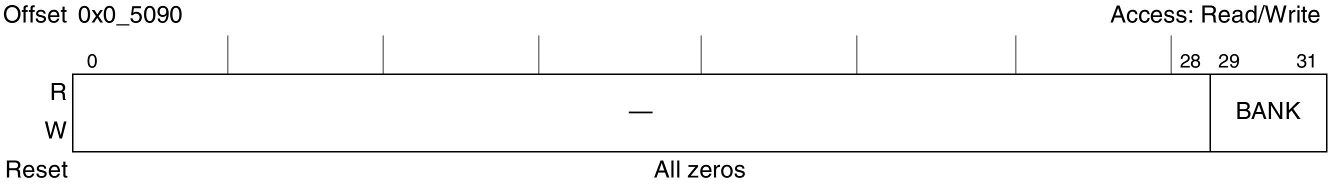


**Figure 13-11. Special Operation Initiation Register (LSOR)**

Table 13-14 describes LSOR.

**Table 13-14. LSOR Field Description**

| Bits | Name | Description |
|---|---|---|
| 0–28 | — | Reserved |
| 29–31 | BANK | Bank on which a special operation is initiated. If the bank identified by BANK is marked valid (BR*n*[V] set) and the bank is controlled by a memory controller whose current mode OP is non-zero—or a special operation—eLBC will request the special operation to be activated on the selected bank when this field is written. Otherwise, writing this field has no effect.<br>000 Bank 0 is triggered for special operation<br>...<br>111 Bank 7 is triggered for special operation |

## 13.3.1.8 UPM Refresh Timer (LURT)

The UPM refresh timer (LURT), shown in Figure 13-12, generates a refresh request for all valid banks that selected a UPM machine and are refresh-enabled (M*x*MR[RFEN] = 1). Each time the timer expires, a qualified bank generates a refresh request using the selected UPM. The qualified banks rotate their requests.

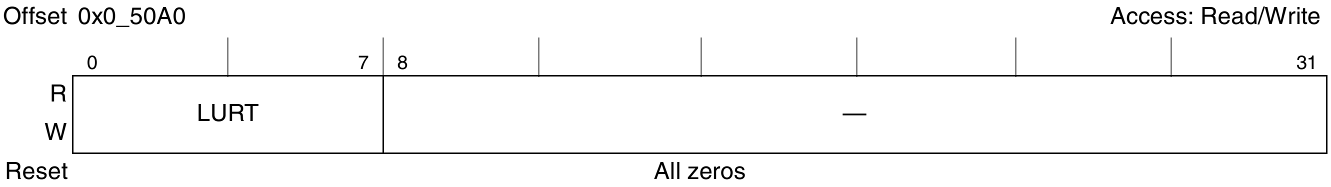


**Figure 13-12. UPM Refresh Timer (LURT)**

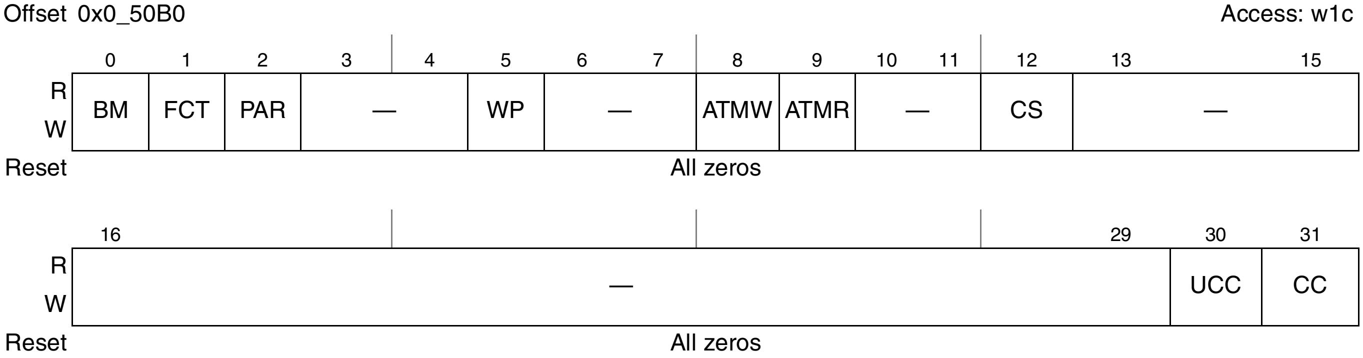
Table 13-15 describes LURT fields.

**Table 13-15. LURT Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–7 | LURT | UPM refresh timer period. Determines, along with the timer prescaler (MRTPR), the timer period according to the following equation:<br>$$\text{TimerPeriod} = \frac{\text{LURT}}{\left(\frac{\text{Fsystemclock}}{\text{MRTPR[PTP]}}\right)}$$<br>Example: For a 266-MHz system clock and a required service rate of 15.6 µs, given MRTPR[PTP] = 32, the LURT value should be 128 decimal. 128/(266 MHz/32) = 15.4 µs, which is less than the required service period of 15.6 µs.<br>Note that the reset value (0x00) sets the maximum period to 256 x MRTPR[PTP] system clock cycles. |
| 8–31 | — | Reserved |

### 13.3.1.9 Transfer Error Status Register (LTESR)

The transfer error status register (LTESR) indicates the cause of an error or event. LTESR, shown in Figure 13-13, is a write-1-to-clear register. Reading LTESR occurs normally; however, write operations can clear but not set bits. A bit is cleared whenever the register is written, and the data in the corresponding bit location is a 1. For example, to clear only the write protect error bit (LTESR[WP]) without affecting other LTESR bits, 0x0400_0000 should be written to the register. After any error/event reported by LTESR, LTEATR[V] must be cleared for LTESR to updated again.



Offset 0x0_50B0 Access: w1c

| | 0 | 1 | 2 | 3–4 | 5 | 6–7 | 8 | 9 | 10–11 | 12 | 13–15 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| R/W | BM | FCT | PAR | — | WP | — | ATMW | ATMR | — | CS | — |
| Reset | All zeros | | | | | | | | | | |

| | 16–29 | 30 | 31 |
|---|---|---|---|
| R/W | — | UCC | CC |
| Reset | All zeros | | |

**Figure 13-13. Transfer Error Status Register (LTESR)**

Table 13-16 describes LTESR fields.

**Table 13-16. LTESR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | BM | Bus monitor time-out<br>0 No local bus monitor time-out occurred.<br>1 Local bus monitor time-out occurred. No data beat was acknowledged on the bus within LBCR[BMT] x LBCR[BMTPS] bus clock cycles from the start of a transaction. |
| 1 | FCT | FCM command time-out<br>0 No FCM command time-out occurred.<br>1 A CW0, CW1, CW2, or CW3 command issued to FCM timed-out with respect to the timer configured by FMR[CWTO]. |
| 2 | PAR | Parity or ECC error<br>0 No local bus parity error<br>1 Local bus parity error (GPCM or UPM), or uncorrectable ECC error (FCM). LTEATR[PB] indicates the byte lane that caused the error and LTEATR[BNK] indicates which memory controller bank was accessed. |
| 3–4 | — | Reserved |
| 5 | WP | Write protect error<br>0 No write protect error occurred.<br>1 A write was attempted to a local bus memory region that was defined as read-only in the memory controller. Usually, in this case, a bus monitor time-out will occur (as the cycle is not automatically terminated). |
| 6–7 | — | Reserved |
| 8 | ATMW | Atomic error write<br>0 No atomic write error occurred.<br>1 The subsequent write (WARA) to a memory bank did not occur within 256 bus clock cycles. |
| 9 | ATMR | Atomic error read<br>0 No atomic read error occurred.<br>1 The subsequent read (RAWA) to a memory bank did not occur within 256 bus clock cycles. |
| 10–11 | — | Reserved |
| 12 | CS | Chip select error<br>0 No chip select error occurred.<br>1 A transaction was sent to the eLBC that did not hit any memory bank. |
| 13–29 | — | Reserved |
| 30 | UCC | UPM Run pattern (MxMR[OP]=11) command completion event<br>0 No UPM Run pattern operation in progress, or operation pending.<br>1 UPM Run pattern operation has completed, allowing software to continue processing of results. |
| 31 | CC | FCM command completion event<br>0 No FCM operation in progress, or operation pending.<br>1 FCM operation has completed, allowing software to continue processing of results. |

### 13.3.1.10 Transfer Error Check Disable Register (LTEDR)

The transfer error check disable register (LTEDR), shown in Figure 13-14, is used to disable error/event checking. Note that control of error/event checking is independent of control of reporting of errors/events (LTEIR) through the interrupt mechanism.

Offset 0x0_50B4 Access: Read/Write

<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td></td><td>15</td></tr>
<tr><td>R<br>W</td><td>BMD</td><td>FCTD</td><td>PARD</td><td colspan="2">—</td><td>WPD</td><td colspan="2">—</td><td>WARA</td><td>RAWA</td><td colspan="2">—</td><td>CSD</td><td colspan="3">—</td></tr>
<tr><td>Reset</td><td colspan="16">All zeros</td></tr>
</table>

<table>
<tr><td></td><td>16</td><td></td><td></td><td>29</td><td>30</td><td>31</td></tr>
<tr><td>R<br>W</td><td colspan="4">—</td><td>UCCD</td><td>CCD</td></tr>
<tr><td>Reset</td><td colspan="6">All zeros</td></tr>
</table>

**Figure 13-14. Transfer Error Check Disable Register (LTEDR)**

Table 13-17 describes LTEDR fields.

**Table 13-17. LTEDR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | BMD | Bus monitor disable<br>0 Bus monitor is enabled.<br>1 Bus monitor is disabled, but internal bus time-outs can still occur. |
| 1 | FCTD | FCM command time-out disable<br>0 FCM command timer is enabled.<br>1 FCM command time-out is disabled, but internal FCM command timer can terminate command waits. |
| 2 | PARD | Parity and ECC error checking disabled. Note that uncorrectable read errors may cause the assertion of *core_fault_in*, which causes the core to generate a machine check interrupt, unless it is disabled.<br>0 Parity and ECC error checking is enabled.<br>1 Parity and ECC error checking is disabled. |
| 3–4 | — | Reserved |
| 5 | WPD | Write protect error checking disable.<br>0 Write protect error checking is enabled.<br>1 Write protect error checking is disabled. |
| 6–7 | — | Reserved |
| 8 | WARA | Write after read atomic (WARA) error checking disable.<br>0 WARA error checking is enabled.<br>1 WARA error checking is disabled. |
| 9 | RAWA | Read after write atomic (RAWA) error checking disable.<br>0 RAWA error checking is enabled.<br>1 RAWA error checking is disabled. |
| 10–11 | — | Reserved |

**Table 13-17. LTEDR Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 12 | CSD | Chip select error checking disable.<br>0 Chip select error checking is enabled.<br>1 Chip select error checking is disabled. |
| 13–29 | — | Reserved |
| 30 | UCCD | UPM Run pattern command completion checking disable.<br>0 UPM Run pattern command completion checking is enabled.<br>1 UPM Run pattern command completion checking is disabled. |
| 31 | CCD | FCM command completion checking disable.<br>0 Command completion checking is enabled.<br>1 Command completion checking is disabled. |

### 13.3.1.11 Transfer Error Interrupt Enable Register (LTEIR)

The transfer error interrupt enable register (LTEIR), shown in Figure 13-15, is used to send or block error/event reporting through the eLBC internal interrupt mechanism. Software should clear pending errors/events in LTESR before enabling interrupts. After an interrupt has occurred, clearing relevant LTESR error/event bits negates the interrupt.

Offset 0x0_50B8 Access: Read/Write

| | 0 | 1 | 2 | 3 4 | 5 | 6 7 | 8 | 9 | 10 11 | 12 | 13 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | BMI | FCTI | PARI | — | WPI | — | WARA | RAWA | — | CSI | — |
| Reset | All zeros | | | | | | | | | | |

| | 16 29 | 30 | 31 |
|---|---|---|---|
| R<br>W | — | UCCI | CCI |
| Reset | All zeros | | |

**Figure 13-15. Transfer Error Interrupt Enable Register (LTEIR)**

Table 13-18 describes LTEIR fields.

**Table 13-18. LTEIR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | BMI | Bus monitor error interrupt enable.<br>0 Bus monitor error reporting is disabled.<br>1 Bus monitor error reporting is enabled. |
| 1 | FCTI | FCM command time-out interrupt enable.<br>0 FCM command time-out error reporting is disabled.<br>1 FCM command time-out error reporting is enabled. |

**Table 13-18. LTEIR Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 2 | PARI | Parity and ECC error interrupt enable. Note that uncorrectable read errors may cause the assertion of core_fault_in, which causes the core to generate a machine check interrupt, unless it is disabled (by clearing HID1[RFXE]). If RFXE is zero and this error occurs, LTEDR[PARD] must be cleared and PARI must be set to ensure that an interrupt is generated.<br>0 Parity and ECC error reporting is disabled.<br>1 Parity and ECC error reporting is enabled. |
| 3–4 | — | Reserved |
| 5 | WPI | Write protect error interrupt enable.<br>0 Write protect error reporting is disabled.<br>1 Write protect error reporting is enabled. |
| 6–7 | — | Reserved |
| 8 | WARA | Write after read atomic (WARA) error interrupt enable.<br>0 WARA error reporting is disabled.<br>1 WARA error reporting is enabled. |
| 9 | RAWA | Read after write atomic (RAWA) error interrupt enable.<br>0 RAWA error reporting is disabled.<br>1 RAWA error reporting is enabled. |
| 10–11 | — | Reserved |
| 12 | CSI | Chip select error interrupt enable.<br>0 Chip select error reporting is disabled.<br>1 Chip select error reporting is enabled. |
| 13–29 | — | Reserved |
| 30 | UCCI | UPM Run pattern command completion Event interrupt enable.<br>0 UPM Run pattern command completion reporting is disabled.<br>1 UPM Run pattern command completion reporting is enabled. |
| 31 | CCI | FCM command completion Event interrupt enable.<br>0 Command completion reporting is disabled.<br>1 Command completion reporting is enabled. |

### 13.3.1.12 Transfer Error Attributes Register (LTEATR)

The transfer error attributes register (LTEATR) captures source attributes of an error/event. Figure 13-16 shows the LTEATR. After LTEATR[V] has been set, software must clear this bit to allow LTESR, LTEATR, and LTEAR to update following any subsequent events/errors.

Offset 0x0_50BC

Access: Read/Write

| | 0 2 | 3 | 4 10 | 11 15 | 16 19 | 20 27 | 28 30 | 31 |
|---|---|---|---|---|---|---|---|---|
| R<br>W | — | RWB | — | SRCID | PB | BNK | — | V |
| Reset | All zeros | | | | | | | |

**Figure 13-16. Transfer Error Attributes Register (LTEATR**

Table 13-19 describes LTEATR fields.

**Table 13-19. LTEATR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–2 | — | Reserved |
| 3 | RWB | Transaction type for the error:<br>0 The transaction for the error was a write transaction.<br>1 The transaction for the error was a read transaction. |
| 4–10 | — | Reserved |
| 11–15 | SRCID | Captures the source of the transaction when this information is provided on the internal interface to the eLBC. |
| 16–19 | PB | Parity error on byte or block. For GPCM and UPM, there are four parity error status bits, one per byte lane. A bit is set for the byte that had a parity error (bit 16 represents byte 0, the most significant byte lane). For FCM, there are at most four 512-byte page blocks (for a large page device) checked by ECC. A bit is set for the 512-byte block that had an uncorrectable ECC error on read (bit 16 represents block 0, the first 512 bytes of a page; if ORx[PGS] = 0, bits 17–19 are always 0). |
| 20–27 | BNK | Memory controller bank. There is one error status bit per memory controller bank (bit 20 represents bank 0). A bit is set for the local bus memory controller bank that had an error. |
| 28–30 | — | Reserved |
| 31 | V | Error attribute capture is valid. Indicates that the captured error information is valid.<br>0 Captured error attributes and address are not valid.<br>1 Captured error attributes and address are valid. |

## 13.3.1.13 Transfer Error Address Register (LTEAR)

The transfer error address register (LTEAR) captures the address of a transaction that caused an error/event. The transfer error address register (LTEAR) is shown in Figure 13-17.

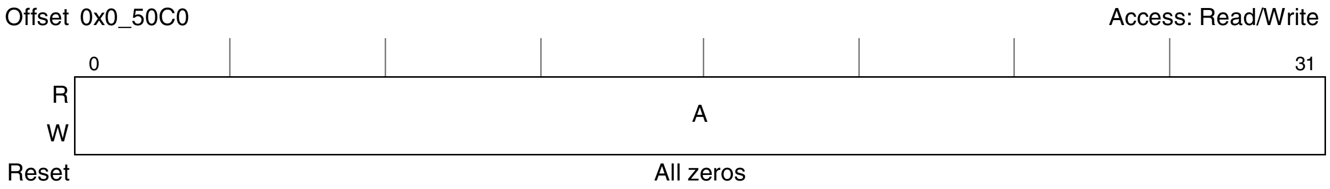


**Figure 13-17. Transfer Error Address Register (LTEAR)**

Table 13-20 describes LTEAR fields.

**Table 13-20. LTEAR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–31 | A | Transaction address for the error. For GPCM and UPM, holds the 32-bit address of the transaction resulting in an error. For FCM, this register is undefined. |

## 13.3.1.14 Transfer Error ECC Register (LTECCR)

The transfer error ECC register (LTECCR) captures single bit and multibit errors per 512-byte sector in FCM mode. LTECCR, shown in Figure 13-18, is a write-1-to-clear register. Write operations can clear but not set bits. It captures the errors during full page read transfers on FCM command completion event, provided ECC check is enabled in BRx[DECC].

Offset 0x0_50C4 Access: w1c

| | 0 11 | 12 15 | 16 27 | 28 31 |
|---|---|---|---|---|
| R<br>W | — | SBCE | — | MBUE |
| Reset | All zeros | | | |

**Figure 13-18. Transfer Error ECC Register (LTECCR)**

**Table 13-21. LTECCR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–11 | — | Reserved |
| 12–15 | SBCE | Single bit correctable error<br>There are at most four 512-byte page blocks (for a large page device) checked by ECC. A bit is set for the 512-byte block that had a single bit correctable ECC error on read (bit 12 represents block 0, the first 512 bytes of a page; if ORx[PGS] = 0, bits 13–15 are always 0). |
| 16–27 | — | Reserved |
| 28–31 | MBUE | Multi bit uncorrectable error<br>There are at most four 512-byte page blocks (for a large page device) checked by ECC. A bit is set for the 512-byte block that had an uncorrectable ECC error on read (bit 28 represents block 0, the first 512 bytes of a page; if ORx[PGS] = 0, bits 29–31 are always 0). |

### 13.3.1.15 Local Bus Configuration Register (LBCR)

The local bus configuration register (LBCR) is shown in Figure 13-19.

Offset 0x0_50D0 Access: Read/Write

| | 0 | 1 7 | 8 9 | 10 | 11 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|
| R<br>W | LDIS | — | BCTLC | AHD | — | LPBSE | EPAR |
| Reset | All zeros | | | | | | |

| | 16 23 | 24 27 | 28 31 |
|---|---|---|---|
| R<br>W | BMT | — | BMTPS |
| Reset | All zeros | | |

**Figure 13-19. Local Bus Configuration Register**

Table 13-22 describes LBCR fields.

**Table 13-22. LBCR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | LDIS | Local bus disable<br>0 Local bus is enabled.<br>1 Local bus is disabled. No internal transactions will be acknowledged. |
| 1–7 | — | Reserved |
| 8–9 | BCTLC | Defines the use of LBCTL<br>00 LBCTL is used as W/$\overline{\text{R}}$ control for GPCM or UPM accesses (buffer control).<br>01 LBCTL is used as $\overline{\text{LOE}}$ for GPCM accesses only.<br>10 LBCTL is used as $\overline{\text{LWE}}$ for GPCM accesses only.<br>11 Reserved. |
| 10 | AHD | Address hold disable. Removes part of the hold time for LAD with respect to LALE in order to lengthen the LALE pulse<br>0 During address phases on the local bus, the LALE signal negates one platform clock period prior to the address being invalidated. For instance, at 33.3 MHz, this provides 3 ns of additional address hold time at the external address latch.<br>1 During address phases on the local bus, the LALE signal negates 0.5 platform clock period prior to the address being invalidated. This halves the address hold time, but extends the latch enable duration. This may be necessary for very high frequency designs. |
| 11–13 | — | Reserved. |
| 14 | LPBSE | Enables parity byte select on $\overline{\text{LGTA}}$/LFR$\overline{\text{B}}$/LGPL4/LUPWAIT/LPBSE signal.<br>0 Parity byte select is disabled. $\overline{\text{LGTA}}$/LGPL4/LPBSE signal is available for memory control as LGPL4 (output) or $\overline{\text{LGTA}}$/LFR$\overline{\text{B}}$/LUPWAIT (input).<br>1 Parity byte select is enabled. LPBSE signal is dedicated as the parity byte select output, and $\overline{\text{LGTA}}$/LFR$\overline{\text{B}}$/LUPWAIT is disabled. |
| 15 | EPAR | Determines odd or even parity. Writing GPCM or UPM controlled memory with EPAR = 1 and reading the memory with EPAR = 0 generates parity errors for testing.<br>0 Odd parity; normal, odd-parity ECC<br>1 Even parity; inverted, even-parity ECC |
| 16–23 | BMT | Bus monitor timing. Defines the bus monitor time-out period. Clearing BMT (reset value) selects the maximum count of bus clock cycles. For non-zero values of BMT, the number of LCLK clock cycles to count down before a time-out error is generated is given by:<br>bus cycles = BMT × PS, where PS is set according to LBCR[BMTPS].<br>The value of BMT × PS must not be less than 40 bus cycles for reliable operation. |

**Table 13-22. LBCR Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 24–27 | — | Reserved |
| 28–31 | BMTPS | Bus monitor timer prescale. Defines the multiplier, PS, to scale LBCR[BMT] for determining bus time-outs.<br>0000 PS = 8<br>0001 PS = 16<br>0010 PS = 32<br>0011 PS = 64<br>0100 PS = 128<br>0101 PS = 256<br>0110 PS = 512<br>0111 PS = 1024<br>1000 PS = 2048<br>1001 PS = 4096<br>1010 PS = 8192<br>1011 PS = 16,384<br>1100 PS = 32,768<br>1101 PS = 65,536<br>1110 PS = 131,072<br>1111 PS = 262,144 |

### 13.3.1.16 Clock Ratio Register (LCRR)

The clock ratio register, shown in Figure 13-20, sets the system clock to eLBC bus frequency ratio. It also provides configuration bits for extra delay cycles for address and control signals.

**NOTE**

> For proper operation of the system, it is required that this register setting will not be altered while local bus memories or devices are being accessed. Special care needs to be taken when running instructions from an eLBC memory.

Offset 0x0_50D4 Access: Read/Write

| | 0 | 1–13 | 14–15 | 16–26 | 27–31 |
|---|---|---|---|---|---|
| R<br>W | PBYP | — | EADC | — | CLKDIV |
| Reset | 1 | 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 0 | 0 0 0 0 0 0 0 0 0 0 0 | 0 *n* *n* 0 0 |

**Figure 13-20. Clock Ratio Register (LCRR)**

Table 13-23 describes LCRR fields.

**Table 13-23. LCRR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | PBYP | PLL bypass. This bit should be set when using low bus clock frequencies (See device hardware specifications for applicable frequencies.). When in PLL bypass mode, incoming data is captured in the middle of the bus clock cycle.<br>0 The PLL is enabled.<br>1 The PLL is bypassed. |
| 1–13 | — | Reserved |
| 14–15 | EADC | External address delay cycles of LCLK. Defines the number of cycles for the assertion of LALE.<br>00 4<br>01 1<br>10 2<br>11 3 |
| 16–26 | — | Reserved |
| 27–31 | CLKDIV | System clock divider. Sets the frequency ratio between the system clock and the local bus clock. The system clock is equivalent to ccb_clk . Only the values shown below are allowed. Note that the reset value of CLKDIV depends on the boot ROM location configuration (cfg_rom_loc[0:3]). If the FCM is selected as the boot ROM location, the default value for CLKDIV is 0100; otherwise, the default is 1000.<br>**Note:** It is critical that no transactions are being executed via the local bus while CLKDIV is being modified. As such, prior to modification, the user must ensure that code is not executing out of the local bus. Once LCRR[CLKDIV] is written, the register should be read, and then an isync should be executed.<br>00000–00001 Reserved<br>00010 4<br>00011 Reserved<br>00100 8 (default for boot ROM set to FCM)<br>00101–00111 Reserved<br>01000 16 (default for boot ROM not set to FCM)<br>01001–11111 Reserved |

### 13.3.1.17 Flash Mode Register (FMR)

The local bus Flash mode register (FMR), shown in Figure 13-21, controls global operation of the FCM.

Offset 0x0_50E0 Access: Read/Write

| | 0–15 | 16–19 | 20 | 21–22 | 23 | 24–25 | 26–27 | 28–29 | 30–31 |
|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | CWTO | BOOT | — | ECCM | — | AL | — | OP |
| Reset | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 0 0 0 | R[1] | 0 0 | 0 | 0 0 | 0 0 | 0 0 | 0 0 |

**Figure 13-21. Flash Mode Register**

[1] Bit R (field BOOT) is set if power-on-reset configuration selects FCM as the boot ROM target.

Table 13-24 describes FMR fields.

**Table 13-24. FMR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–15 | — | Reserved |
| 16–19 | CWTO | Command wait time-out. For FCM commands that wait on $\overline{\text{LFRB}}$ being sampled high (CW0, CW1, RBW and RSW), FCM pauses execution of the instruction sequence until either $\overline{\text{LFRB}}$ is sampled high, or a timer controlled by CTO expires, whichever occurs first. The time-out in the latter case is:<br>0000 256 cycles of LCLK<br>0001 512 cycles of LCLK<br>0010 1024 cycles of LCLK<br>0011 2048 cycles of LCLK<br>0100 4096 cycles of LCLK<br>0101 8192 cycles of LCLK<br>0110 16,384 cycles of LCLK<br>0111 32,768 cycles of LCLK<br>1000 65,536 cycles of LCLK<br>1001 131,072 cycles of LCLK<br>1010 262,144 cycles of LCLK<br>1011 524,288 cycles of LCLK<br>1100 1,048,576 cycles of LCLK<br>1101 2,097,152 cycles of LCLK<br>1110 4,194,304 cycles of LCLK<br>1111 8,388,608 cycles of LCLK |
| 20 | BOOT | Flash auto-boot load mode. During system boot from NAND Flash EEPROM, this bit remains set to alter the use of the FCM buffer RAM. Software should clear BOOT once FCM is to be restored to normal operation. Setting BOOT without auto-boot in progress only alters the mapping of the buffer RAM.<br>0 FCM is operating in normal functional mode, with an 8 Kbyte FCM buffer RAM.<br>1 eLBC has been configured—either from reset or by a special operation OP = 01—to auto-load a 4-Kbyte boot block into the FCM buffer RAM, which maps only the 4 Kbytes of NAND flash main data region comprising the boot block. Any access to the buffer RAM is delayed until the entire boot block has been loaded. |
| 21–22 | — | Reserved |
| 23 | ECCM | ECC mode. When hardware checking and/or generation of error correcting codes (ECC) is enabled (that is, when BR*n*[DECC] is 01 or 10, and full page transfers are specified with FBCR[BC] = 0), ECCM sets the ECC block size and position of the ECC code word(s) in the NAND Flash spare region for both checking and generation functions. The format of the ECC code word conforms with the Samsung/Toshiba spare region assignment specifications.<br>0 ECC is checked/calculated over 512-Byte blocks. A 24-bit ECC is assigned to spare region bytes at offsets ($N$×16)+6 through ($N$×16)+8 for spare region $N$, $N$ = 0–3.<br>1 ECC is checked/calculated over 512-Byte blocks. A 24-bit ECC is assigned to spare region bytes at offsets ($N$×16)+8 through ($N$×16)+10 for spare region $N$, $N$ = 0–3. |
| 24–25 | — | Reserved |

**Table 13-24. FMR Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 26–27 | AL | Address length. AL sets the number of address bytes issued during page address (PA) operations. However, the number of address bytes issued for column address (CA) operations is determined by the device page size (for OR*n*[PGS] = 0, 1 CA byte is issued; for OR*n*[PGS] = 1, 2 CA bytes are issued).<br>00 2 bytes are issued for page addresses, thus a total of 3 (OR*n*[PGS] = 0) or 4 (OR*n*[PGS] = 1) address bytes are issued for a {CA,PA} sequence<br>01 3 bytes are issued for page addresses, thus a total of 4 (OR*n*[PGS] = 0) or 5 (OR*n*[PGS] = 1) address bytes are issued for a {CA,PA} sequence<br>10 4 bytes are issued for page addresses, thus a total of 5 (OR*n*[PGS] = 0) or 6 (OR*n*[PGS] = 1) address bytes are issued for a {CA,PA} sequence<br>11 — |
| 28–29 | — | Reserved |
| 30–31 | OP | Flash operation. For OP not equal to 00, a special operation is triggered on the next write to LSOR or dummy access to a bank controlled by FCM. Once a special operation has commenced, OP is automatically reset to 00 by FCM. Individual blocks may be temporarily unlocked for erase and reprogramming operations.<br>00 Normal operation. All read and write accesses to banks controlled by FCM access the shared FCM buffer RAM. No bus activity is caused by this operation.<br>01 Simulate auto-boot block loading, and set FMR[BOOT]. Boot block loading occurs from the bank triggered on the special operation, therefore the appropriate bank configuration must be initialized prior to issuing this operation.<br>10 Execute the command sequence contained in FIR, but with write protection enabled (pin $\overline{\text{LFWP}}$ asserted low) so that all Flash blocks are protected from accidental erasure and reprogramming.<br>11 Execute the command sequence contained in FIR, but permit the single block identified by FBAR[BLK] to be erased or reprogrammed, with pin $\overline{\text{LFWP}}$ remaining high during the access. |

### 13.3.1.18 Flash Instruction Register (FIR)

The local bus Flash instruction register (FIR), shown in Figure 13-22, holds a sequence of up to eight instructions for issue by the FCM. Setting FMR[OP] non-zero and writing LSOR or accessing a bank controlled by FCM causes FCM to read FIR 4 bits at a time, starting at bit 0 and continuing with adjacent 4-bit opcodes, until only NOP opcodes remain. The programmed instruction sequence of OP0, OP1,..., OP7 is performed on the activated bank, using the data buffer addressed by FPAR. If LTEIR[CCI] = 1 and LTEDR[CCD] = 0, eLBC will generate an interrupt once the entire sequence has completed, and software should examine LTEATR and clear its V bit.

Software must not alter the contents of the addressed FCM buffer, FIR, MDR, FCR, FBAR, FPAR, or FBCR while an operation is in progress—or eLBC will behave unpredictably—but software can freely modify the contents of any currently unused FCM RAM buffer in preparation for the next operation.

Offset 0x0_50E4 Access: Read/Write

| | 0 3 | 4 7 | 8 11 | 12 15 | 16 19 | 20 23 | 24 27 | 28 31 |
|---|---|---|---|---|---|---|---|---|
| R<br>W | OP0 | OP1 | OP2 | OP3 | OP4 | OP5 | OP6 | OP7 |
| Reset | All zeros | | | | | | | |

**Figure 13-22. Flash Instruction Register**

Table 13-25 describes FIR fields.

**Table 13-25. FIR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–3 | OP0 | FCM operation codes. OP0 is executed first, followed by OP1, through to OP7.<br>0000 NOP—No-operation and end of operation sequence<br>0001 CA—Issue current column address as set in FPAR, with length set by ORx[PGS]<br>0010 PA—Issue current block+page address as set in FBAR and FPAR, with length set by FMR[AL]<br>0011 UA—Issue user-defined address byte from next AS field in MDR<br>0100 CM0—Issue command from FCR[CMD0]<br>0101 CM1—Issue command from FCR[CMD1]<br>0110 CM2—Issue command from FCR[CMD2]<br>0111 CM3—Issue command from FCR[CMD3]<br>1000 WB—Write FBCR bytes of data from current FCM buffer to Flash device<br>1001 WS—Write one byte (8b port) of data from next AS field of MDR to Flash device<br>1010 RB—Read FBCR bytes of data from Flash device into current FCM RAM buffer<br>1011 RS—Read one byte (8b port) of data from Flash device into next AS field of MDR<br>1100 CW0—Wait for LFR$\overline{\text{B}}$ to return high or time-out, then issue command from FCR[CMD0]<br>1101 CW1—Wait for LFR$\overline{\text{B}}$ to return high or time-out, then issue command from FCR[CMD1]<br>1110 RBW—Wait for LFR$\overline{\text{B}}$ to return high or time-out, then read FBCR bytes of data from Flash device into current FCM RAM buffer<br>1111 RSW—Wait for LFR$\overline{\text{B}}$ to return high or time-out, then read one byte (8b port) of data from Flash device into next AS field of MDR |
| 4–7 | OP1 | |
| 8–11 | OP2 | |
| 12–15 | OP3 | |
| 16–19 | OP4 | |
| 20–23 | OP5 | |
| 24–27 | OP6 | |
| 28–31 | OP7 | |

### 13.3.1.19 Flash Command Register (FCR)

The local bus Flash command register (FCR), shown in Figure 13-23, holds up to four NAND Flash EEPROM command bytes that may be referenced by opcodes in FIR during FCM operation. The values of the commands should follow the manufacturer’s datasheet for the relevant NAND Flash device.

Offset 0x0_50E8 Access: Read/Write

| | 0 … 7 | 8 … 15 | 16 … 23 | 24 … 31 |
|---|---|---|---|---|
| R<br>W | CMD0 | CMD1 | CMD2 | CMD3 |
| Reset | All zeros | | | |

**Figure 13-23. Flash Command Register**

Table 13-26 describes FCR fields.

**Table 13-26. FCR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–7 | CMD0 | General purpose FCM Flash command byte 0. Opcodes in FIR that issue command index 0 write CMD0 to the NAND Flash command/data bus. |
| 8–15 | CMD1 | General purpose FCM Flash command byte 1. Opcodes in FIR that issue command index 1 write CMD1 to the NAND Flash command/data bus. |
| 16–23 | CMD2 | General purpose FCM Flash command byte 2. Opcodes in FIR that issue command index 2 write CMD2 to the NAND Flash command/data bus. |
| 24–31 | CMD3 | General purpose FCM Flash command byte 3. Opcodes in FIR that issue command index 3 write CMD3 to the NAND Flash command/data bus. |

### 13.3.1.20 Flash Block Address Register (FBAR)

The local bus Flash block address register (FBAR), shown in Figure 13-24, locates the NAND Flash block index for the page currently accessed.

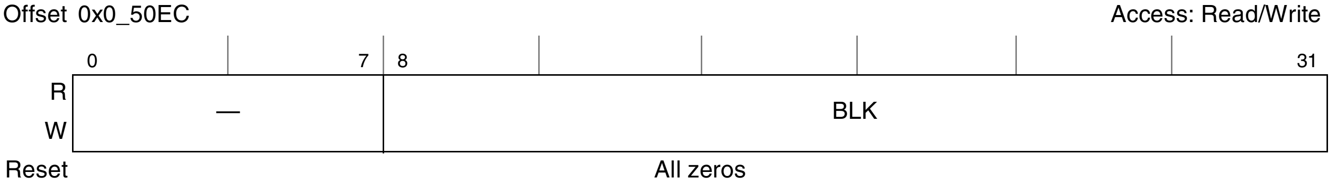


**Figure 13-24. Flash Block Address Register**

Table 13-27 describes FBAR fields.

**Table 13-27. FBAR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–7 | — | Reserved |
| 8–31 | BLK | Flash block address. The size of the NAND Flash, as configured in OR*n*[PGS] and FMR[AL], determines the number of bits of BLK that are issued to the EEPROM during block address phases. |

### 13.3.1.21 Flash Page Address Register (FPAR)

The local bus Flash page address register (FPAR), shown in Figure 13-25 and Figure 13-26, locates the current NAND Flash page in both the external NAND Flash device and FCM buffer RAM.

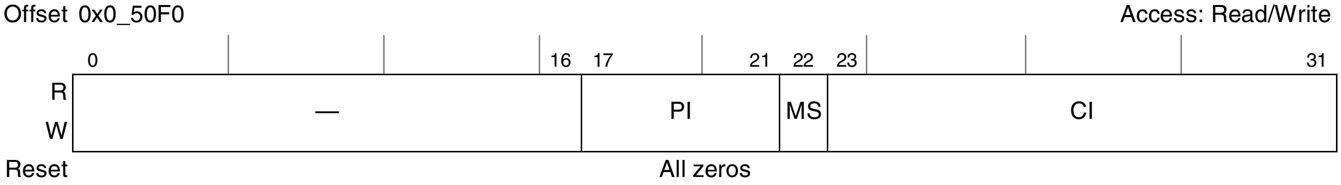


**Figure 13-25. Flash Page Address Register, Small Page Device (ORx[PGS] = 0)**

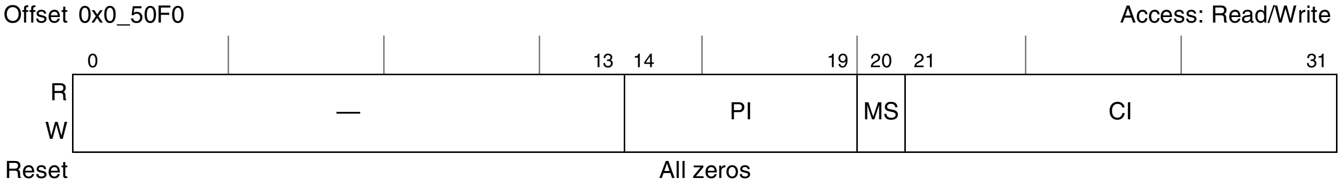


**Figure 13-26. Flash Page Address Register, Large Page Device (ORx[PGS] = 1)**

Table 13-28 describes FPAR fields for small page devices.

**Table 13-28. FPAR Field Descriptions, Small Page Device (ORx[PGS] = 0)**

| Bits | Name | Description |
|---|---|---|
| 0–16 | — | Reserved |
| 17–21 | PI | Page index. PI indexes the page in NAND Flash EEPROM at the current block defined by FBAR, and locates the corresponding transfer buffer in the FCM buffer RAM.<br>The 3 LSBs of PI index one of the eight 1 Kbyte buffers in the FCM buffer RAM as follows:<br>000 The page is transferred to/from FCM buffer 0, address offsets 0x0000–0x03FF<br>001 The page is transferred to/from FCM buffer 1, address offsets 0x0400–0x07FF<br>010 The page is transferred to/from FCM buffer 2, address offsets 0x0800–0x0BFF<br>011 The page is transferred to/from FCM buffer 3, address offsets 0x0C00–0x0FFF<br>100 The page is transferred to/from FCM buffer 4, address offsets 0x1000–0x13FF<br>101 The page is transferred to/from FCM buffer 5, address offsets 0x1400–0x17FF<br>110 The page is transferred to/from FCM buffer 6, address offsets 0x1800–0x1BFF<br>111 The page is transferred to/from FCM buffer 7, address offsets 0x1C00–0x1FFF |
| 22 | MS | Main/spare region locator. In the case that FBCR[BC] = 0, MS is treated as 0.<br>0 Data is transferred to/from the main region of the FCM buffer; that is, the first 512 bytes of the buffer are used as the starting address.<br>1 Data is transferred to/from the spare region of the FCM buffer; that is, the second 512 bytes of the buffer are used as the starting address, but only an initial 16 bytes of spare region are defined. |
| 23–31 | CI | Column index. CI indexes the first byte to transfer to/from the main or spare region of the NAND Flash EEPROM and corresponding transfer buffer. In the case that FBCR[BC] = 0, CI is treated as 0. For MS = 0, CI can range 0x000–0x1FF; for MS = 1, CI can range 0x000–0x00F. |

Table 13-29 describes FPAR fields for large page devices.

**Table 13-29. FPAR Field Descriptions, Large Page Device (ORx[PGS] = 1)**

| Bits | Name | Description |
|---|---|---|
| 0–13 | — | Reserved |
| 14–19 | PI | Page index. PA indexes the page in NAND Flash EEPROM at the current block defined by FBAR, and locates the corresponding transfer buffer in the FCM buffer RAM.<br>The LSB of PI indexes one of the two 4 Kbyte buffers in the FCM buffer RAM as follows:<br>0 The page is transferred to/from FCM buffer 0, address offsets 0x0000–0x0FFF<br>1 The page is transferred to/from FCM buffer 1, address offsets 0x1000–0x1FFF |
| 20 | MS | Main/spare region locator. In the case that FBCR[BC] = 0, MS is treated as 0.<br>0 Data is transferred to/from the main region of the FCM buffer; that is, the first 2048 bytes of the buffer are used as the starting address.<br>1 Data is transferred to/from the spare region of the FCM buffer; that is, the second 2048 bytes of the buffer are used as the starting address, but only an initial 64 bytes of spare region are defined. |
| 21–31 | CI | Column index. CI indexes the first byte to transfer to/from the main or spare region of the NAND Flash EEPROM and corresponding transfer buffer. In the case that FBCR[BC] = 0, CI is treated as 0. For MS = 0, CI can range 0x000–0x7FF; for MS = 1, CI can range 0x000–0x03F. |

### 13.3.1.22 Flash Byte Count Register (FBCR)

The local bus Flash byte count register (FBCR), shown in Figure 13-27, defines the size of FCM block transfers for reads and writes to the NAND Flash EEPROM.

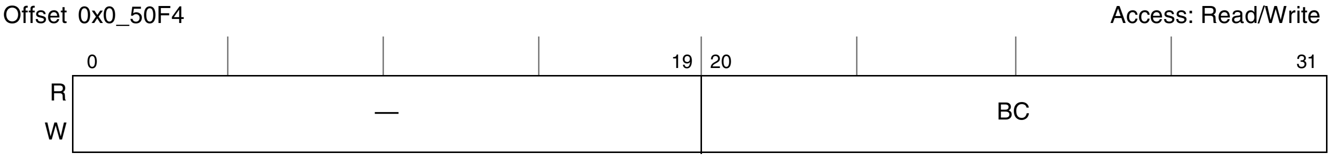


**Figure 13-27. Flash Byte Count Register**

Table 13-30 describes FBCR fields.

**Table 13-30. FBCR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–19 | — | Reserved |
| 20–31 | BC | Byte count determines how many bytes are transferred by the FCM during data read (RB) or data write (WB) opcodes.<br>The first byte accessed in the NAND Flash EEPROM is located by the FPAR register, and successive bytes are transferred until either BC bytes have been counted, or the end of the spare region of the currently addressed Flash page has been reached.<br>If BC = 0, an entire Flash page and its spare region will be transferred by FCM, in which case FPAR[MS] and FPAR[CI] are treated as zero regardless of their values. BC = 0 is the only setting that permits FCM to generate and check ECC. |

### 13.3.1.23 Flash ECC Block*n* Register (FECC0–FECC3)

The local bus flash ECC block*n* register (FECC*n*), shown in Figure 13-28, specifies the ECC value calculated during writes or reads by eLBC. It can be used for verify after write feature in software. Note that the valid bit sets before the command completion event and hence the correct ECC could be read before actual completion of writes/reads.

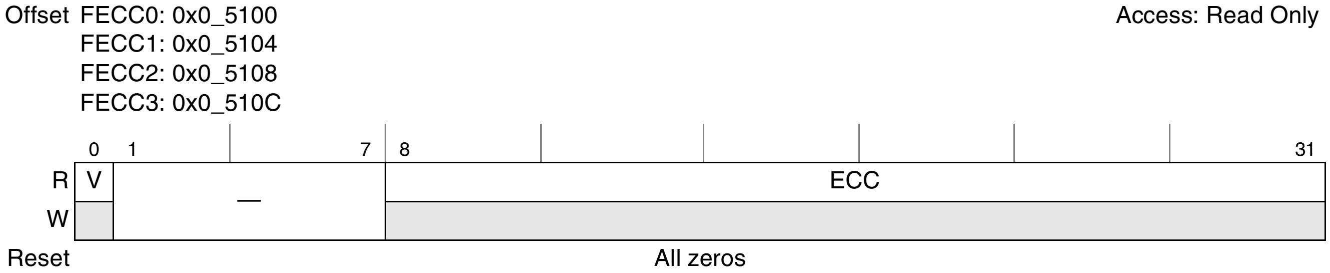


**Figure 13-28. Flash ECC Block*n* Register (FECC0–FECC3)**

Table 13-31. FECC*n* Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0 | V | Valid bit. This bit denotes that the ECC stored in this register is valid. It is set for full page write/read transfers if ECC generation/checking is enabled in BR*n*[DECC]. |
| 1–7 | — | Reserved |
| 8–31 | ECC | 24 bit ECC; For $n^{th}$ 512 bytes of a page in case of large page or for $(4k + n)^{th}$ 512 byte page for small page where $k = 0,1,2,...$). It stores calculated ECC value during writes/reads. |

## 13.4 Functional Description

The eLBC allows the implementation of memory systems with very specific timing requirements.

- The GPCM provides interfacing for simpler, lower-performance memories and memory-mapped devices. It has inherently lower performance because it does not support bursting. For this reason, GPCM-controlled banks are used primarily for boot-loading from NVRAM or NOR Flash, and access to low-performance memory-mapped peripherals.
- The FCM interfaces the eLBC to NAND Flash EEPROMs with 8-bit data bus. The FCM has an automatic boot-loading feature that allows the CPU to boot from high density EEPROM, loading the boot block into 4 Kbytes of RAM for execution of the first level boot code. Following boot, FCM provides a flexible instruction sequencer that allows a user-defined command, address, and data transfer sequence of up to 8 steps to be executed against a memory-mapped buffer RAM. Programmable set-up time, hold time, and wait states permit the FCM to maximize the performance of NAND Flash block transfers, which can proceed in parallel with software processing of the multiple RAM buffers. A single-pass ECC engine in the FCM permits zero-overhead error checking, reporting, and correction in both boot blocks and page data transfers if enabled.
- The UPM supports refresh timers, address multiplexing of the external bus, and generation of programmable control signals for row address and column address strobes, to allow for a minimal glue logic interface to DRAMs, burstable SRAMs, and almost any other kind of peripheral with asynchronous timing or single data rate clocking. The UPM can be used to generate flexible, user-defined timing patterns for control signals that govern a memory device. These patterns define how the external control signals behave during a read, write, burst-read, or burst-write access. Refresh timers are also available to periodically initiate user-defined refresh patterns.

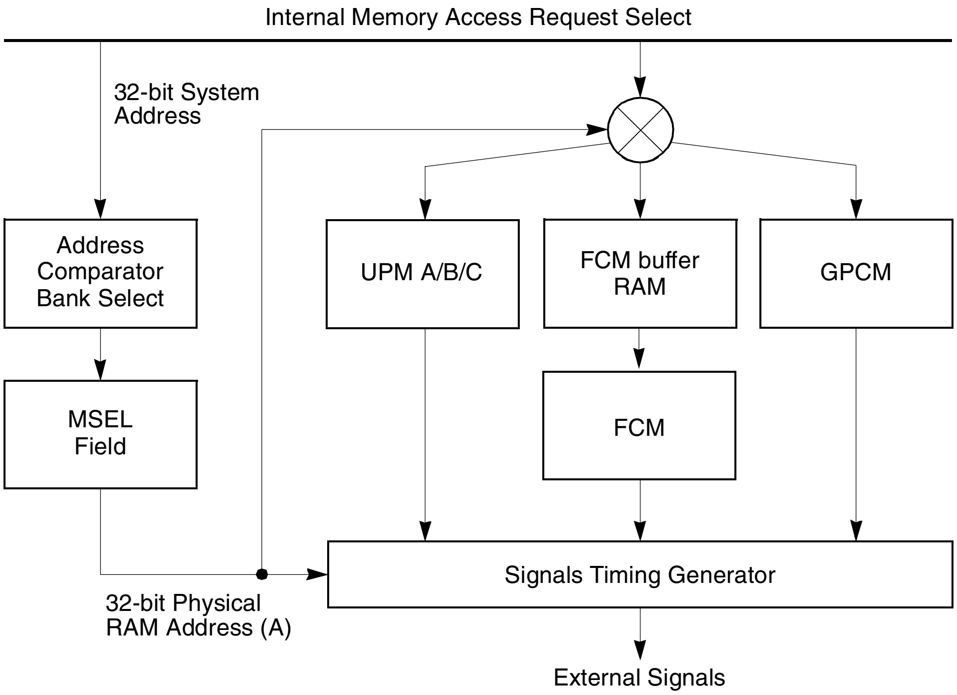


**Figure 13-29. Basic Operation of Memory Controllers in the eLBC**

Each memory bank (chip select) can be assigned to any of these three types of machines through the machine select bits of the base register for that bank (BR*n*[MSEL]), as illustrated in Figure 13-29. If a bank match occurs, the corresponding machine (GPCM, FCM, or UPM) then takes ownership of the external signals that control the access and maintains control until the transaction ends.

## 13.4.1 Basic Architecture

The following subsections describe the basic architecture of the eLBC.

### 13.4.1.1 Address and Address Space Checking

The defined base addresses are written to the BR*n* registers, while the corresponding address masks are written to the OR*n* registers. Each time a local bus access is requested, the internal transaction address is compared with each bank. Addresses are decoded by comparing the 17 MSBs of the address, masked by OR*n*[AM], with the base address for each bank (BR*n*[BA]). If a match is found on a memory controller bank, the attributes defined in the BR*n* and OR*n* for that bank are used to control the memory access. If a match is found in more than one bank, the lowest-numbered bank handles the memory access (that is, bank 0 has priority over bank 1).

### 13.4.1.2 External Address Latch Enable Signal (LALE)

The local bus uses a multiplexed address/data bus. Therefore the eLBC must distinguish between address and data phases, which take place on the same bus (LAD pins). The LALE signal, when asserted, signifies an address phase during which the eLBC drives the memory address on the LAD pins. An external address latch uses this signal to capture the address and provide it to the address pins of the memory or peripheral device. When LALE is negated, LAD then serves as the (bi-directional) data bus for the access. Any address phase initiates the assertion of LALE, which has a programmable duration of between 1 and 4 bus clock cycles.

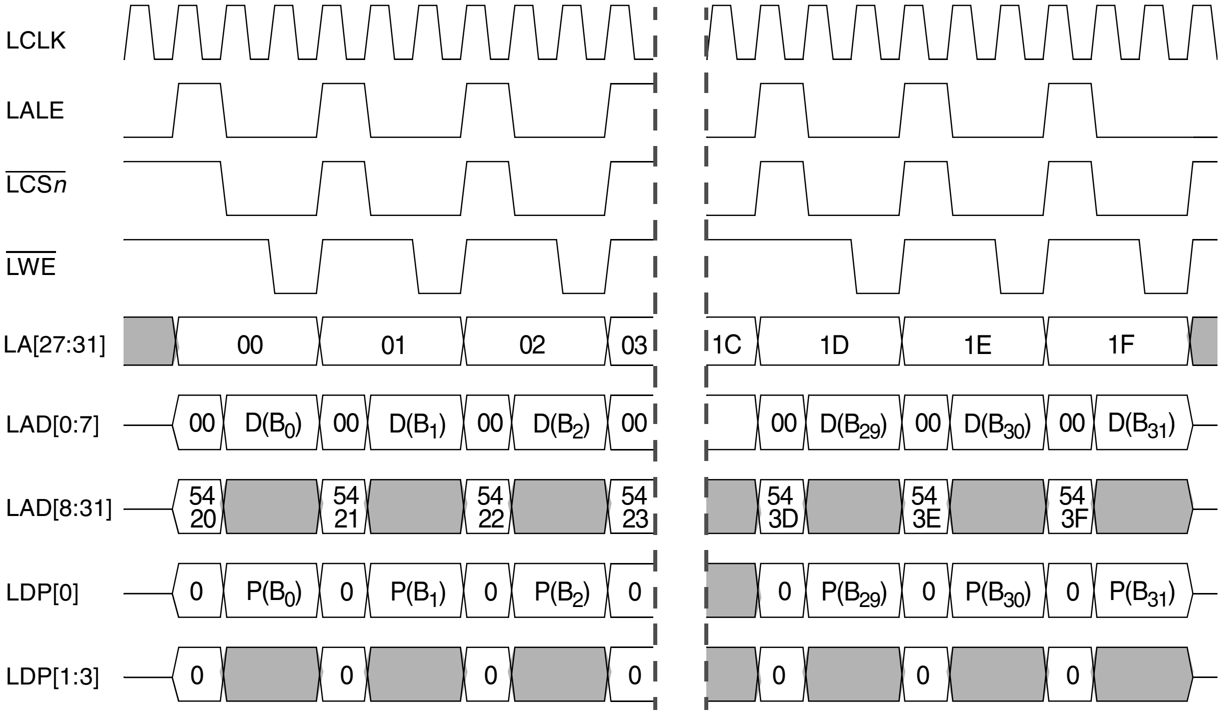
To ensure adequate hold time on the external address latch, LALE negates earlier than the address changes on LAD during address phases. By default, LALE negates earlier by 1 platform clock period. For example, if the platform clock is operating at 533 MHz, then 1.8 ns of address hold time is introduced. However, at higher frequencies, the duration of the shortened LALE pulse may not meet the minimum latch enable pulse width specifications of some latches. In such cases, setting LBCR[AHD] = 1 increases the LALE pulse width by ½ platform clock cycle, but decreases the address hold time by the same amount. If both longer hold time and longer LALE pulse duration are needed, then the address phase can be extended using the ORn[EAD] and LCRR[EADC] fields, and the LBCR[AHD] bit can be left at 0. However, this will add latency to all address tenures.

The frequency of LALE assertion varies across the three memory controllers:

- For GPCM, every assertion of $\overline{\mathrm{LCS}n}$ is considered an independent access, and accordingly, LALE asserts prior to each such access. For example, GPCM driving an 8-bit port would assert LALE and $\overline{\mathrm{LCS}n}$ 32 times in order to satisfy a 32-byte cache line transfer.
- For FCM, LALE asserts prior to each multi-command operation sequence, but LALE can be ignored on NAND Flash EEPROM accesses as the signal does *not* enable address latching in such devices. The value on the LAD and LA pins during LALE assertion is driven low-impedance, but otherwise not defined for FCM banks.
- In the case of UPM, the frequency of LALE assertion depends on how the UPM RAM is programmed. UPM single accesses typically assert LALE once, upon commencement, but it is possible to program UPM to assert LALE several times, and to change the values of LA*n* with and without LALE being involved.

In general, when using the GPCM controller it is not necessary to use LA if a sufficiently wide latch is used to capture the entire address during LALE phases. The UPMs may require LA if the eLBC is generating its own burst address sequence.

To illustrate how a large transaction is handled by the eLBC, Figure 13-30 shows eLBC signals for the GPCM performing a 32-byte write starting at address 0x5420. Note that during each of the 32 assertions of LALE, LA[27:31] exactly mirror LAD[27:31], but during data phases, only LAD[0:7] and LDP[0] are driven with valid data and parity, respectively.



**Note**: All address and signal values are shown in hexadecimal.
D(Bk) = k$^{th}$ of 32 data bytes, P(Bk) = parity bit of k$^{th}$ data byte.

**Figure 13-30. Example of 8-Bit GPCM Writing 32 Bytes to Address 0x5420 (LCRR[PBYP] = 0)**

### 13.4.1.3 Data Transfer Acknowledge (TA)

The three memory controllers in the eLBC generate an internal transfer acknowledge signal, TA, to allow data on LAD to be either sampled (for reads) or changed (on writes). The data sampling/data change always occurs at the end of the bus cycle in which the eLBC asserts TA internally. In eLBC debug mode, TA is also visible externally on the MDVAL pin. The GPCM controller automatically generates TA according to the timing parameters programmed for them in the option and mode registers; FCM generates TA whenever data read and write instructions are executed out of register FIR; a UPM generates TA only when a UPM pattern has the UTA RAM word bit set. Figure 13-31 shows LALE, TA (internal), and LCS*n*. Note that TA and LALE are never asserted together, and that for the duration of LALE, LCS*n* (or any other control signal) remains negated or frozen.

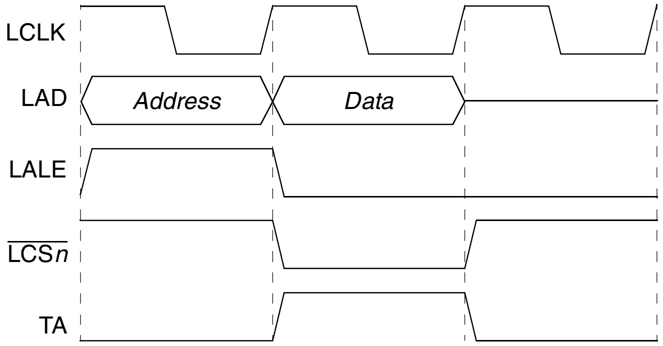


**Figure 13-31. Basic eLBC Bus Cycle with LALE, TA, and $\overline{\text{LCS}n}$**

### 13.4.1.4 Data Buffer Control (LBCTL)

The memory controller provides a data buffer control signal for the local bus (LBCTL). This signal is activated when a GPCM-, FCM-, or UPM-controlled bank is accessed. LBCTL can be disabled by setting OR*n*[BCTLD]. LBCTL can be further configured by LBCR[BCTLC] to act as an extra $\overline{\text{LWE}}$ or an extra $\overline{\text{LOE}}$ signal when in GPCM mode.

If LBCTL is configured as a data buffer control (LBCR[BCTLC] = 00), the signal is asserted (high) on the rising edge of the bus clock on the first cycle of the memory controller operation, coincident with LALE. If the access is a write, LBCTL remains high for the whole duration. However, if the access is a read, LBCTL is negated (low) with the negation of LALE so that the memory device is able to drive the bus. If back-to-back read accesses are pending, LBCTL is asserted (high) one bus clock cycle before the next transaction starts (that is, one bus clock cycle before LALE) to allow a whole bus cycle for the bus to turn around before the next address is driven.

### 13.4.1.5 Atomic Operation

The eLBC supports the following kinds of atomic bus operations (set by BR*n*[ATOM]):

- Read-after-write atomic (RAWA). When a write access hits a memory bank in which ATOM = 01, the eLBC reserves the selected memory bank for the exclusive use of the accessing master.

  While the bank is reserved, no other device can be granted access to this bank. The reservation is released when the master that created it accesses the same bank with a read transaction. Additional write transactions prior to the releasing read do not change reservation status, but are otherwise processed normally. If the master fails to release the reservation within 256 bus clock cycles, the reservation is released and an atomic error is reported (if enabled); additional write transactions prior to the releasing read restart the reservation timer. This feature is intended for CAM operations.
- Write-after-read atomic (WARA). When a read access hits a memory bank in which ATOM = 10, the eLBC reserves the bus for the exclusive use of the accessing master.

  During the reservation period, no other device can be granted access to the atomic bank. The reservation is released when the device that created it accesses the same bank with a write transaction. Additional read transactions prior to the releasing write are otherwise processed normally and do not change the reservation status. If the device fails to release the reservation within 256 bus clock cycles, the reservation is released and an atomic error is reported (if enabled); additional read transactions prior to the releasing write restart the reservation timer.

#### 13.4.1.6 Parity Generation and Checking (LDP)

Parity can be configured for any GPCM or UPM bank by programming BR*n*[DECC]. Parity is generated and checked on a per-byte basis using LDP[0:3] for the bank if BR*n*[DECC] = 01 (normal parity) or BR*n*[DECC] = 10 for read-modify-write (RMW) parity. Byte lane parity on LDP[0:3] is generated regardless of the BR*n*[DECC] setting. Note that RMW parity can be used only for 32-bit port size banks. LBCR[EPAR] determines the global type of parity (odd or even).

FCM calculates an ECC over 512-byte blocks, and hence does not use the LDP[0:3] pins. The setting of BR*n*[DECC] = 01 enables ECC checking only, while BR*n*[DECC] = 10 enables ECC generation and checking; in either case, LBCR[EPAR] determines the global type of block parity for ECC (odd or even).

#### 13.4.1.7 Bus Monitor

A bus monitor is provided to ensure that each bus cycle is terminated within a reasonable (user defined) period. When a transaction starts, the bus monitor starts counting down from the time-out value (LBCR[BMT] × LBCR[BMTPS]) until a data beat is acknowledged on the bus. It then reloads the time-out value and resumes the countdown until the data tenure completes and then idles if there is no pending transaction. Setting LTEDR[BMD] disables bus monitor error checking (i.e. the LTESR[BM] bit is not set by a bus monitor time-out); however, the bus monitor is still active and can generate a UPM exception (as noted in Section 13.4.4.1.4, “Exception Requests,”) or terminate a GPCM access.

It is very important to ensure that the value of LBCR[BMT] is not set too low; otherwise spurious bus time-outs may occur during normal operation—resulting in incomplete data transfers. Accordingly, the time-out value represented by the LBCR[BMT], LBCR[BMTPS] pair must not be set below 40 bus cycles for time-out under any circumstances.

#### 13.4.1.8 PLL Bypass Mode

At LCLK frequencies in excess of 66 MHz the local bus PLL is used to provide improved hold times at external receivers, and ease set-up margins for read data captured by eLBC. A wire loop between pins LSYNC_OUT and LSYNC_IN establishes the amount of LCLK skewing achieved by the PLL, which locks so as to produce edges on LCLK before the transition of other eLBC control and data signals.

At lower frequencies, the PLL may be unable to lock or provide sufficient hold time improvement for particularly slow devices. Accordingly, LCRR[PBYP] should be set to 1 to bypass the PLL at low frequencies, with the eLBC generating LCLK directly, while skewing it by half a bus clock cycle. An illustration of GPCM or UPM timing both with and without the PLL activated are shown in Figure 13-32 and Figure 13-33. When LCRR[PBYP] = 0, the skew, $t_{LSKEW}$, matches the round-trip propagation delay of the timing loop between LSYNC_OUT and LSYNC_IN, and data is generated or sampled on the next rising edge of LCLK. The timing diagrams shown normally in this chapter assume that LCRR[PBYP] = 0. When LCRR[PBYP] = 1, the skew equals half the period of LCLK to maximize hold time at the external receiver; in this bypass mode, eLBC drives new address, data, and control signals effectively on falling edges of LCLK, but continues to sample synchronous read data on rising edges of LCLK to maximize the set-up margin for reads.

**NOTE**

Since LCLK is not used for NAND Flash EEPROMs controlled by FCM, the eLBC drives and samples data on the same edge (rising edge when LCRR[PBYP] = 0 and falling edge when LCRR[PBYP] = 1) on FCM controlled banks.

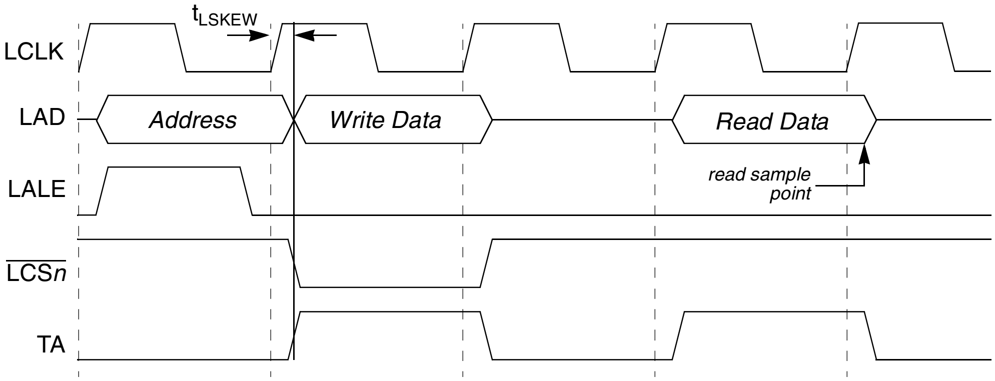


**Figure 13-32. eLBC Bus Cycles in PLL Mode (GPCM and UPM only)**

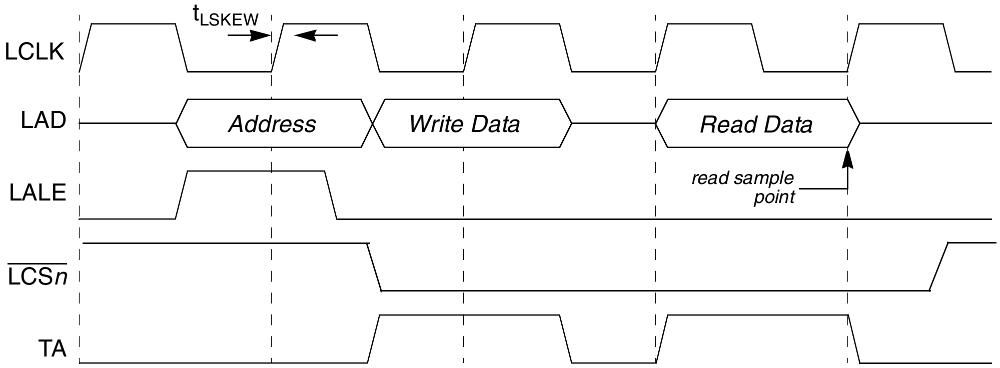


**Figure 13-33. eLBC Bus Cycles in PLL-bypassed Mode (GPCM and UPM only)**

## 13.4.2 General-Purpose Chip-Select Machine (GPCM)

The GPCM allows a minimal glue logic and flexible interface to SRAM, EPROM, FEPROM, ROM devices, and external peripherals. The GPCM contains two basic configuration register groups—BR*n* and OR*n*.

Figure 13-34 shows a simple connection between an 8-bit port size SRAM device and the eLBC in GPCM mode. Byte-write enable signals ($\overline{\text{LWE}}$) are available for each byte written to memory. Also, the output enable signal ($\overline{\text{LOE}}$) is provided to minimize external glue logic. On system reset, a global (boot) chip-select is available that provides a boot ROM chip-select ($\overline{\text{LCS0}}$) prior to the system being fully configured.

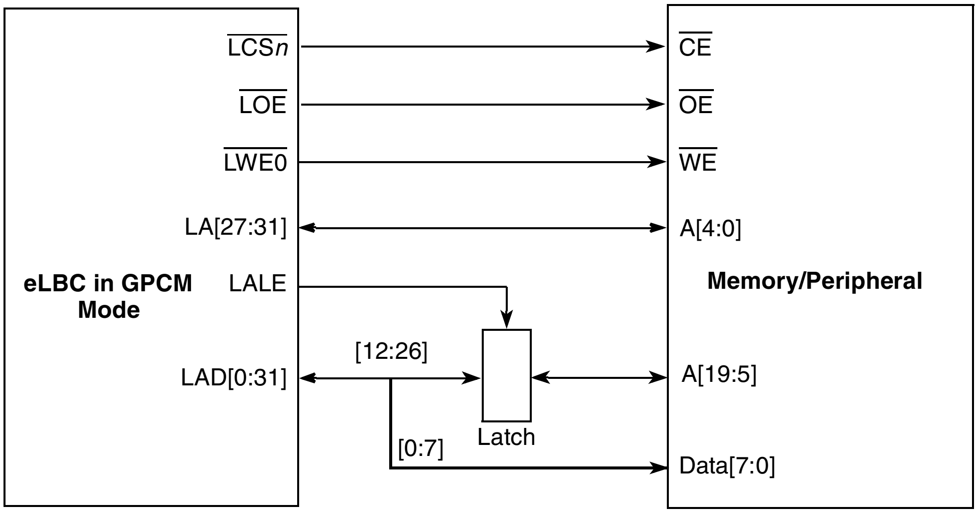


**Figure 13-34. Enhanced Local Bus to GPCM Device Interface**

Figure 13-35 shows $\overline{\text{LCS}}$ as defined by the setup time required between the address lines and $\overline{\text{CE}}$. The user can configure OR*n*[ACS] to specify $\overline{\text{LCS}}$ to meet this requirement. Generally, the attributes for the memory cycle are taken from OR*n*. These attributes include the CSNT, ACS, XACS, SCY, TRLX, EHTR and SETA fields.

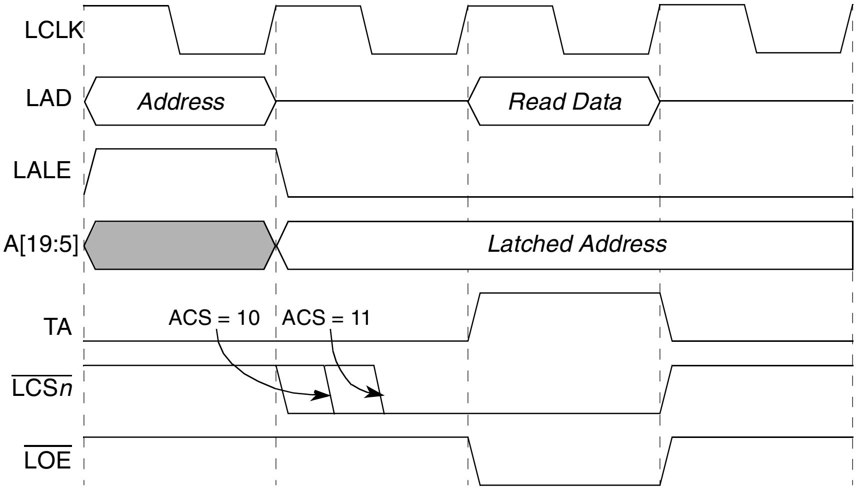
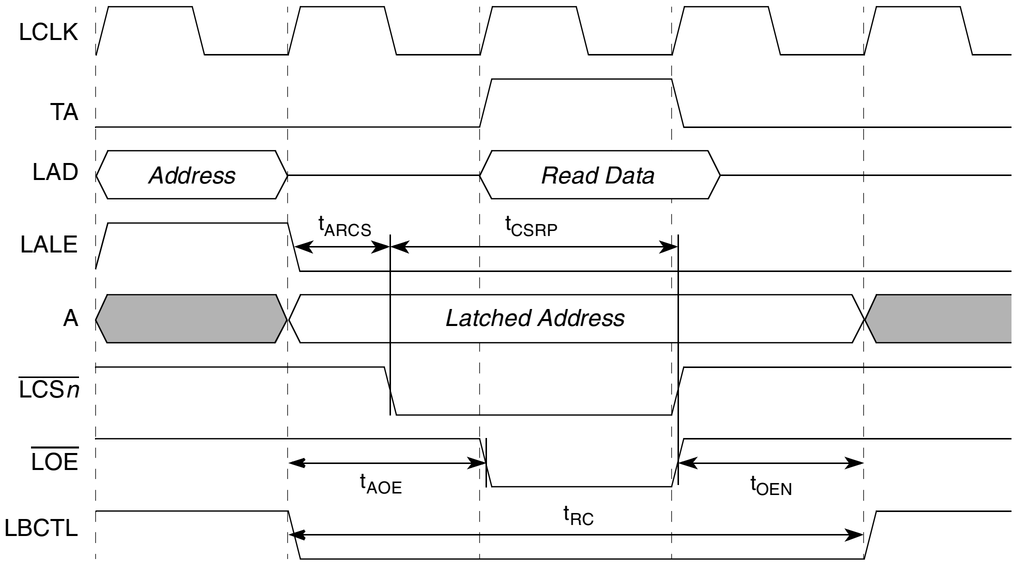


**Figure 13-35. GPCM Basic Read Timing (XACS = 0, ACS = 1x, TRLX = 0)**

#### 13.4.2.1 GPCM Read Signal Timing

The basic GPCM read timing parameters that may be set by the OR*n* attributes are shown in Figure 13-36. The read access cycle commences upon latching of the memory address (LALE negated), and concludes when LBCTL returns high to turn the local bus around for a subsequent address phase. Read data is captured by eLBC on the falling edge of TA. $\overline{\text{LOE}}$ and $\overline{\text{LCS}n}$ negate high simultaneously, in some cases before the end of the read access to provide additional hold time for the external memory.



**Notes:**
$t_{RC}$ = Read cycle time.
$t_{ARCS}$ = Address valid to read chip-select time.
$t_{AOE}$ = Address valid to output enable time.
$t_{CSRP}$ = Read chip-select assertion period.
$t_{OEN}$ = Output enable negated time.

**Figure 13-36. GPCM General Read Timing Parameters**

Table 13-32 lists the signal timing parameters for a GPCM read access as the option register attributes are varied.

**Table 13-32. GPCM Read Control Signal Timing**

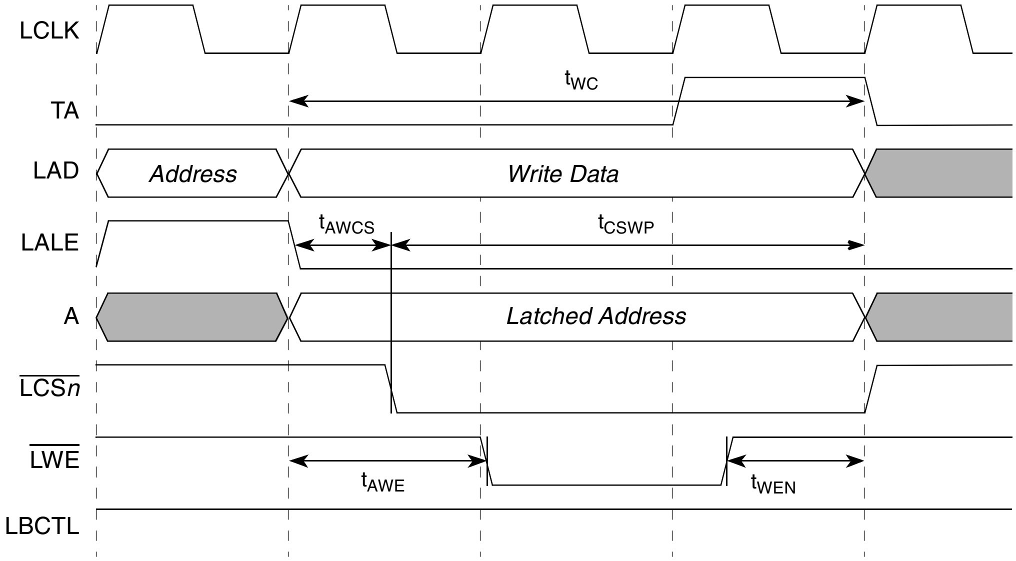
| Option Register Attributes | | | | Signal Timing (LCLK clock cycles) | | | | |
|---|---|---|---|---|---|---|---|---|
| TRLX | EHTR | XACS | ACS | $t_{ARCS}$ | $t_{CSRP}$ | $t_{AOE}$ | $t_{OEN}$ | $t_{RC}$ |
| 0 | 0 | 0 | 0X | 0 | 2+SCY | 1 | 0 | 2+SCY |
| 0 | 0 | 0 | 10 | ¼ | 1¾+SCY | 1 | 0 | 2+SCY |
| 0 | 0 | 0 | 11 | ½ | 1½+SCY | 1 | 0 | 2+SCY |
| 0 | 0 | 1 | 0X | 0 | 2+SCY | 1 | 0 | 2+SCY |
| 0 | 0 | 1 | 10 | 1 | 1+SCY | 1 | 0 | 2+SCY |
| 0 | 0 | 1 | 11 | 2 | 1+SCY | 2 | 0 | 3+SCY |
| 0 | 1 | 0 | 0X | 0 | 2+SCY | 1 | 1 | 3+SCY |
| 0 | 1 | 0 | 10 | ¼ | 1¾+SCY | 1 | 1 | 3+SCY |
| 0 | 1 | 0 | 11 | ½ | 1½+SCY | 1 | 1 | 3+SCY |
| 0 | 1 | 1 | 0X | 0 | 2+SCY | 1 | 1 | 3+SCY |
| 0 | 1 | 1 | 10 | 1 | 1+SCY | 1 | 1 | 3+SCY |
| 0 | 1 | 1 | 11 | 2 | 1+SCY | 2 | 1 | 4+SCY |
| 1 | 0 | 0 | 0X | 0 | 2+2×SCY | 1 | 4 | 6+2×SCY |
| 1 | 0 | 0 | 10 | 1¼ | 1¾+2×SCY | 2 | 4 | 7+2×SCY |

Table 13-32. GPCM Read Control Signal Timing (continued)

| Option Register Attributes | | | | Signal Timing (LCLK clock cycles) | | | | |
|---|---|---|---|---|---|---|---|---|
| TRLX | EHTR | XACS | ACS | $t_{ARCS}$ | $t_{CSRP}$ | $t_{AOE}$ | $t_{OEN}$ | $t_{RC}$ |
| 1 | 0 | 0 | 11 | 1½ | 1½+2×SCY | 2 | 4 | 7+2×SCY |
| 1 | 0 | 1 | 0X | 0 | 2+2×SCY | 1 | 4 | 6+2×SCY |
| 1 | 0 | 1 | 10 | 2 | 1+2×SCY | 2 | 4 | 7+2×SCY |
| 1 | 0 | 1 | 11 | 3 | 1+2×SCY | 3 | 4 | 8+2×SCY |
| 1 | 1 | 0 | 0X | 0 | 2+2×SCY | 1 | 8 | 10+2×SCY |
| 1 | 1 | 0 | 10 | 1¼ | 1¾+2×SCY | 2 | 8 | 11+2×SCY |
| 1 | 1 | 0 | 11 | 1½ | 1½+2×SCY | 2 | 8 | 11+2×SCY |
| 1 | 1 | 1 | 0X | 0 | 2+2×SCY | 1 | 8 | 10+2×SCY |
| 1 | 1 | 1 | 10 | 2 | 1+2×SCY | 2 | 8 | 11+2×SCY |
| 1 | 1 | 1 | 11 | 3 | 1+2×SCY | 3 | 8 | 12+2×SCY |

### 13.4.2.2 GPCM Write Signal Timing

The basic GPCM write timing parameters that may be set by the OR*n* attributes are shown in Figure 13-37. The write access cycle commences upon latching of the memory address (LALE negated), and concludes when $\overline{\text{LCS}n}$ returns high. LBCTL remains stable for the entire cycle to drive data onto any secondary data bus. Write data becomes invalid following the falling edge of TA. $\overline{\text{LWE}}$ may, in some cases, negate high before the end of the write access to provide additional hold time for the external memory.



**Notes:**

$t_{WC}$ = Write cycle time.

$t_{AWCS}$ = Address valid to write chip-select time.

$t_{AWE}$ = Address valid to write enable time.

$t_{CSWP}$ = Write chip-select assertion period.

$t_{WEN}$ = Write enable negated time wrt chip-sele

Figure 13-37. GPCM General Write Timing Parameters

Table 13-33 lists the signal timing parameters for a GPCM write access as the option register attributes are varied.

Table 13-33. GPCM Write Control Signal Timing

| Option Register Attributes | | | | Signal Timing (LCLK clock cycles) | | | | |
|---|---|---|---|---|---|---|---|---|
| TRLX | XACS | ACS | CSNT | $t_{AWCS}$ | $t_{CSWP}$ | $t_{AWE}$ | $t_{WEN}$ | $t_{WC}$ |
| 0 | 0 | 00 | 0 | 0 | 2+SCY | 1 | 0 | 2+SCY |
| 0 | 0 | 10 | 0 | ¼ | 1¾+SCY | 1 | 0 | 2+SCY |
| 0 | 0 | 11 | 0 | ½ | 1½+SCY | 1 | 0 | 2+SCY |
| 0 | 1 | 00 | 0 | 0 | 2+SCY | 1 | 0 | 2+SCY |
| 0 | 1 | 10 | 0 | 1 | 1+SCY | 1 | 0 | 2+SCY |
| 0 | 1 | 11 | 0 | 2 | 1+SCY | 2 | 0 | 3+SCY |
| 0 | 0 | 00 | 1 | 0 | 2+SCY | 1 | ¼ | 2+SCY |
| 0 | 0 | 10 | 1 | ¼ | 1½+SCY | 1 | 0 | 1¾+SCY |
| 0 | 0 | 11 | 1 | ½ | 1¼+SCY | 1 | 0 | 1¾+SCY |
| 0 | 1 | 00 | 1 | 0 | 2+SCY | 1 | ¼ | 2+SCY |
| 0 | 1 | 10 | 1 | 1 | ¾+SCY | 1 | 0 | 1¾+SCY |
| 0 | 1 | 11 | 1 | 2 | ¾+SCY | 2 | 0 | 2¾+SCY |
| 1 | 0 | 00 | 0 | 0 | 2+2×SCY | 1 | 0 | 2+2×SCY |
| 1 | 0 | 10 | 0 | 1¼ | 1¾+2×SCY | 2 | 0 | 3+2×SCY |
| 1 | 0 | 11 | 0 | 1½ | 1½+2×SCY | 2 | 0 | 3+2×SCY |
| 1 | 1 | 00 | 0 | 0 | 2+2×SCY | 1 | 0 | 2+2×SCY |
| 1 | 1 | 10 | 0 | 2 | 1+2×SCY | 2 | 0 | 3+2×SCY |
| 1 | 1 | 11 | 0 | 3 | 1+2×SCY | 3 | 0 | 4+2×SCY |
| 1 | 0 | 00 | 1 | 0 | 3+2×SCY | 1 | 1¼ | 3+2×SCY |
| 1 | 0 | 10 | 1 | 1¼ | 1½+2×SCY | 2 | 0 | 2¾+2×SCY |
| 1 | 0 | 11 | 1 | 1½ | 1¼+2×SCY | 2 | 0 | 2¾+2×SCY |
| 1 | 1 | 00 | 1 | 0 | 3+2×SCY | 1 | 1¼ | 3+2×SCY |
| 1 | 1 | 10 | 1 | 2 | ¾+2×SCY | 2 | 0 | 2¾+2×SCY |
| 1 | 1 | 11 | 1 | 3 | ¾+2×SCY | 3 | 0 | 3¾+2×SCY |

#### 13.4.2.3 Chip-Select Assertion Timing

The banks selected to work with the GPCM support an option to drive the $\overline{\text{LCS}n}$ signal with different timings (with respect to the external address/data bus). $\overline{\text{LCS}n}$ can be driven in any of the following ways:

- Simultaneous with the latched memory address. (This refers to the externally latched address and not the address timing on LAD. That is, the chip select does not assert during LALE).

- One quarter of a clock cycle later.
- One half of a clock cycle later.
- One clock cycle later (for LCRR[CLKDIV] = 2 (clock ratio of 4)), when OR*n*[XACS] = 1.
- Two clock cycles later, when OR*n*[XACS] = 1.
- Three clock cycles later, when OR*n*[XACS] = 1 and OR*n*[TRLX] = 1.

The timing diagram in Figure 13-35 shows two chip-select assertion timings.

#### 13.4.2.3.1 Programmable Wait State Configuration

The GPCM supports internal generation of transfer acknowledge. It allows between zero and 30 wait states to be added to an access by programming OR*n*[SCY] and OR*n*[TRLX]. Internal generation of transfer acknowledge is enabled if OR*n*[SETA] = 0. If LGTA is asserted externally two bus clock cycles or more before the wait state counter has expired (to allow for synchronization latency), the current memory cycle is terminated by LGTA; otherwise it is terminated by the expiration of the wait state counter. Regardless of the setting of OR*n*[SETA], wait states prolong the assertion duration of both LOE and LWE*n* in the same manner. When TRLX = 1, the number of wait states inserted by the memory controller is doubled from OR*n*[SCY] cycles to 2×OR*n*[SCY] cycles, allowing a maximum of 30 wait states.

#### 13.4.2.3.2 Chip-Select and Write Enable Negation Timing

Figure 13-34 shows a basic connection between the local bus and a static memory device. In this case, LCS*n* is connected directly to CE of the memory device. The LWE[0:3] signals are connected to the respective WE[3:0] signals on the memory device where each LWE[0:3] signal corresponds to a different data byte.

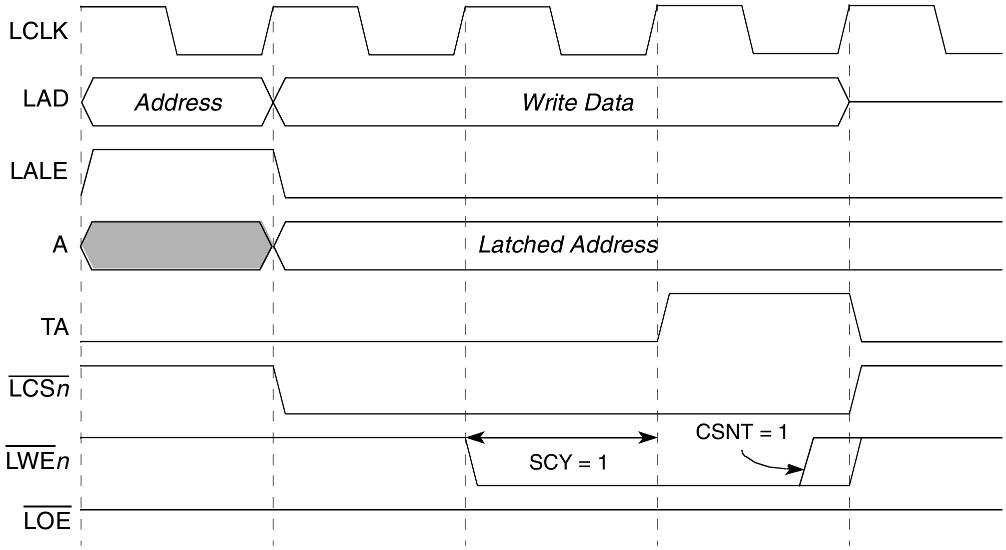


**Figure 13-38. GPCM Basic Write Timing (XACS = 0, ACS = 00, CSNT = 1, SCY = 1, TRLX = 0)**

As Figure 13-38 shows, the timing for LCS*n* is the same as for the latched address. The strobes for the transaction are supplied by LOE or LWE*n*, depending on the transaction direction—read or write (write case shown in the figure). OR*n*[CSNT], along with OR*n*[TRLX], control the timing for the appropriate strobe negation in write cycles. When this attribute is asserted, the strobe is negated one quarter of a clock before the normal case. For example, when ACS = 00 and CSNT = 1, $\overline{\mathrm{LWE}}n$ is negated one quarter of a clock earlier, as shown in Figure 13-38.

1. $\overline{\mathrm{LCS}}n$ is affected by CSNT and TRLX only if ACS[0] is non zero. However, $\overline{\mathrm{LWE}}n$ is affected independent of ACS.
2. When CSNT attribute is asserted, the strobe is negated one quarter of a clock before the normal case.
3. TRLX = 1 in conjunction with CSNT = 1, negates the $\overline{\mathrm{LCS}}n$ and $\overline{\mathrm{LWE}}n$ 1+1/4 cycle earlier.

For example, when ACS = 00, CSNT = 1 and TRLX = 0, $\overline{\mathrm{LWE}}n$ is negated one quarter of a clock earlier and $\overline{\mathrm{LCS}}n$ is negated normally as shown in Figure 13-38.

### 13.4.2.3.3 Relaxed Timing

OR*x*[TRLX] is provided for memory systems that require more relaxed timing between signals. Setting TRLX = 1 has the following effect on timing:

- An additional bus cycle is added between the address and control signals (but only if ACS is not equal to 00).
- The number of wait states specified by SCY is doubled, providing up to 30 wait states.
- The extended hold time on read accesses (EHTR) is extended further.
- $\overline{\mathrm{LCS}}n$ signals are negated one cycle earlier during writes (but only if ACS is not equal to 00).
- $\overline{\mathrm{LWE}}$[0:3] signals are negated one cycle earlier during writes.

Figure 13-39 and Figure 13-40 show relaxed timing read and write transactions. The example in Figure 13-40 also shows address and data multiplexing on LAD for a pair of writes issued consecutively.

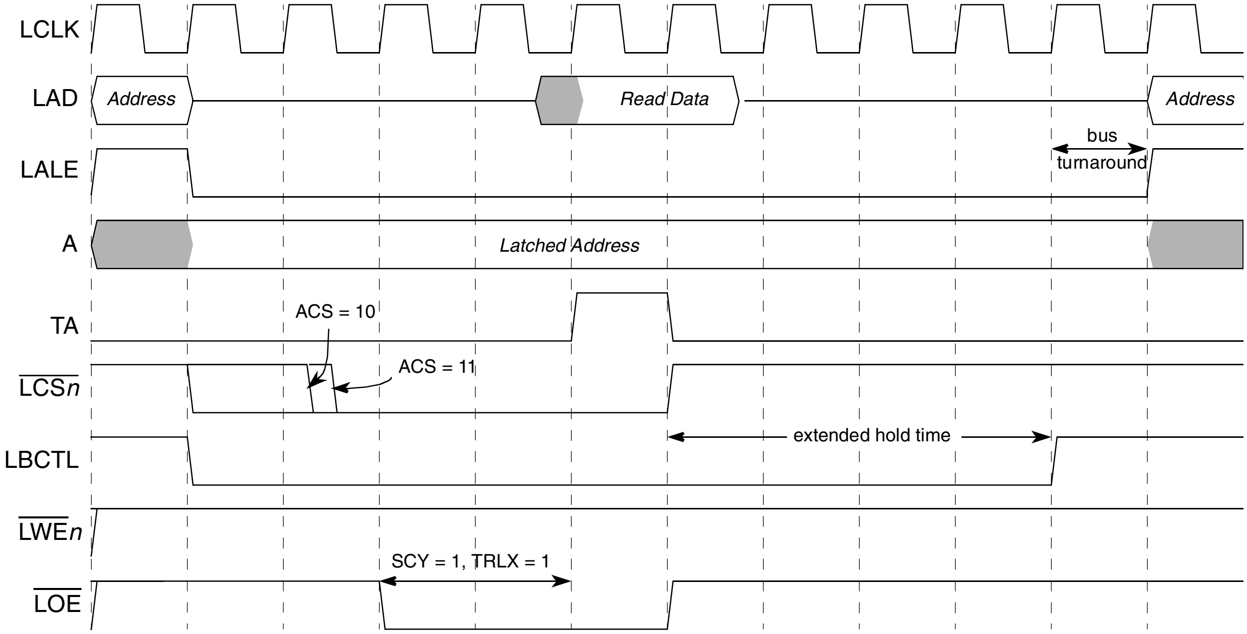


**Figure 13-39. GPCM Relaxed Timing Back-to-Back Reads (XACS = 0, ACS = 1x, SCY = 1, CSNT = 0, TRLX = 1, EHTR = 0)**

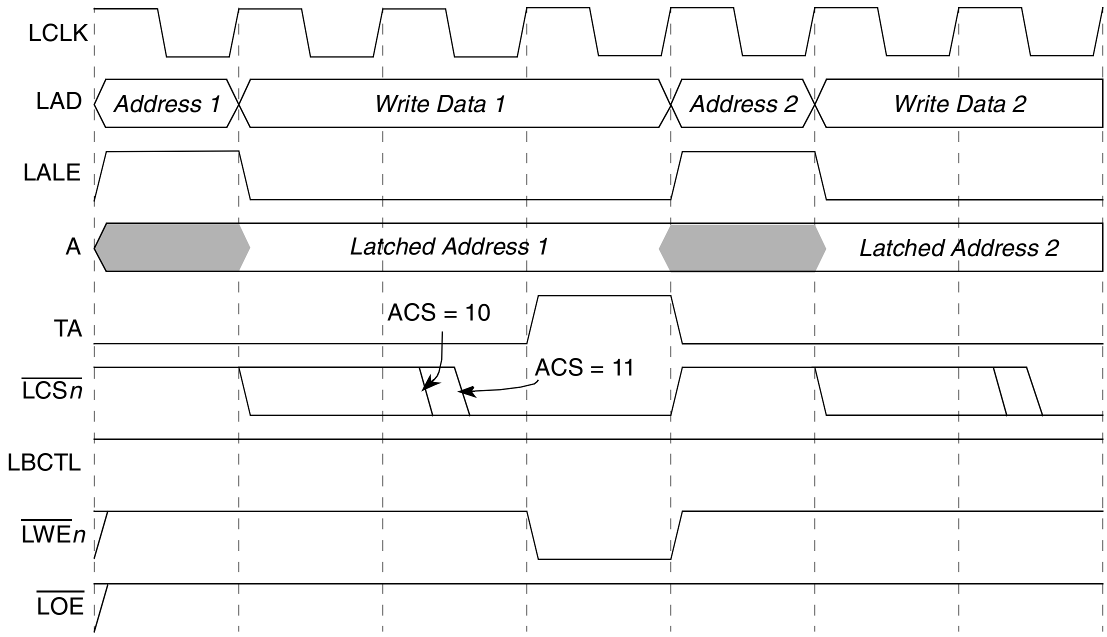


**Figure 13-40. GPCM Relaxed Timing Back-to-Back Writes (XACS = 0, ACS = 1x, SCY = 0, CSNT = 0, TRLX = 1)**

When TRLX and CSNT are set in a write access, the $\overline{\text{LWE}}$[0:3] strobe signals are negated one clock earlier than in the normal case, as shown in Figure 13-41 and Figure 13-42. If ACS ≠ 00, $\overline{\text{LCS}n}$ is also negated one clock earlier.

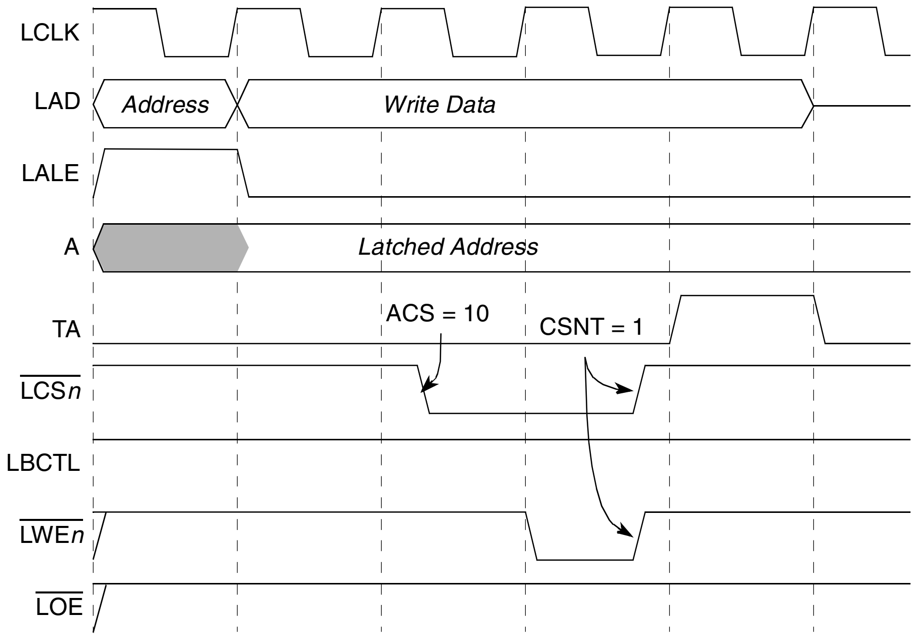


**Figure 13-41. GPCM Relaxed Timing Write (XACS = 0, ACS = 10, SCY = 0, CSNT = 1, TRLX = 1)**

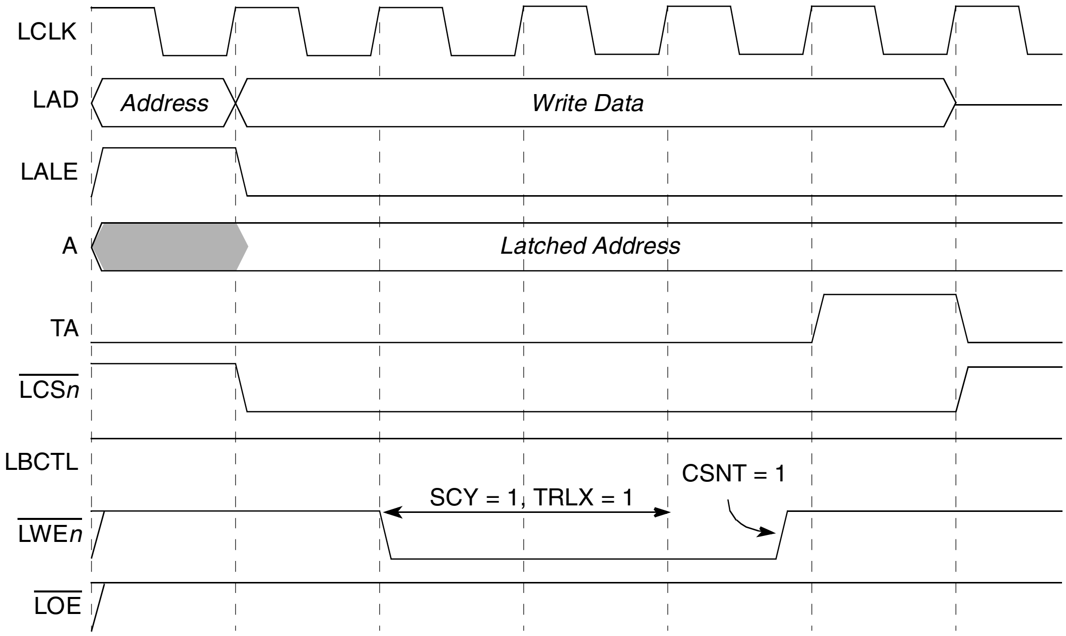


**Figure 13-42. GPCM Relaxed Timing Write (XACS = 0, ACS = 00, SCY = 1, CSNT = 1, TRLX = 1)**

### 13.4.2.3.4 Output Enable (LOE) Timing

The timing of the LOE is affected only by TRLX. It always asserts and negates on the rising edge of the bus clock. LOE asserts either on the rising edge of the bus clock after LCS*n* is asserted or coinciding with LCS*n* (if XACS = 1 and ACS = 10 or ACS = 11). Accordingly, assertion of LOE can be delayed (along with the assertion of LCS*n*) by programming TRLX = 1. LOE negates on the rising clock edge coinciding with LCS*n* negation

### 13.4.2.3.5 Extended Hold Time on Read Accesses

Slow memory devices that take a long time to disable their data bus drivers on read accesses should choose some combination of OR*n*[TRLX,EHTR]. Any access following a read access to the slower memory bank is delayed by the number of clock cycles specified in Table 13-7 in addition to any existing bus turnaround cycle. The final bus turnaround cycle is automatically inserted by the eLBC for reads, regardless of the setting of OR*n*[EHTR].

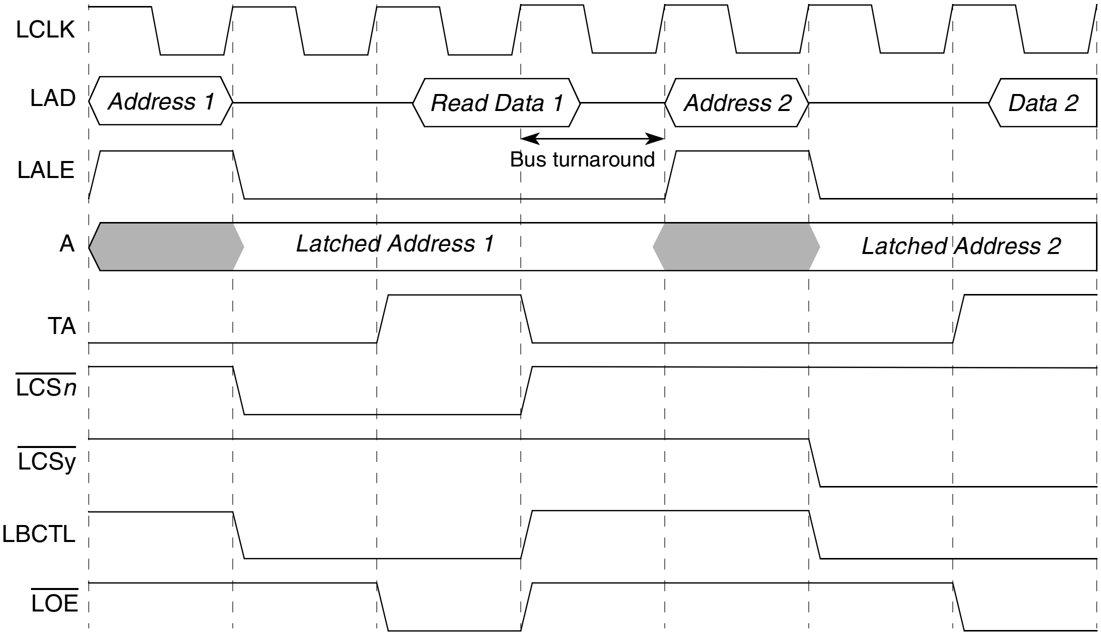


**Figure 13-43. GPCM Read Followed by Read (TRLX = 0, EHTR = 0, Fastest Timing)**

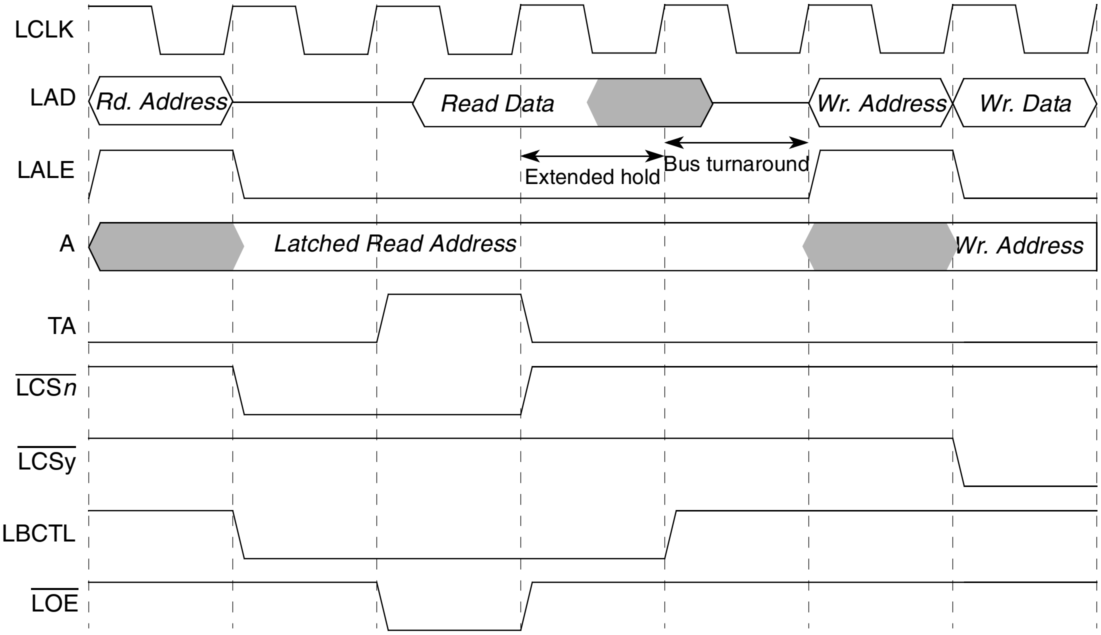


**Figure 13-44. GPCM Read Followed by Write (TRLX = 0, EHTR = 1, One-Cycle Extended Hold Time on Reads)**

### 13.4.2.4 External Access Termination ($\overline{\text{LGTA}}$)

External access termination is supported by the GPCM using the asynchronous $\overline{\text{LGTA}}$ input signal, which is synchronized and sampled internally by the local bus. If, during assertion of $\overline{\text{LCS}n}$, the sampled $\overline{\text{LGTA}}$ signal is asserted, it is converted to an internal generation of transfer acknowledge, which terminates the current GPCM access (regardless of the setting of OR*n*[SETA]). $\overline{\text{LGTA}}$ should be asserted for at least one bus cycle to be effective. Note that because $\overline{\text{LGTA}}$ is synchronized, bus termination occurs two cycles after $\overline{\text{LGTA}}$ assertion, so in case of read cycle, the device still must drive data as long as $\overline{\text{LOE}}$ is asserted.

The user selects whether transfer acknowledge is generated internally or externally ($\overline{\text{LGTA}}$) by programming OR*n*[SETA]. Asserting $\overline{\text{LGTA}}$ always terminates an access, even if OR*n*[SETA] = 0 (internal transfer acknowledge generation), but it is the only means by which an access can be terminated if OR*n*[SETA] = 1. The timing of $\overline{\text{LGTA}}$ is illustrated by the example in Figure 13-45.

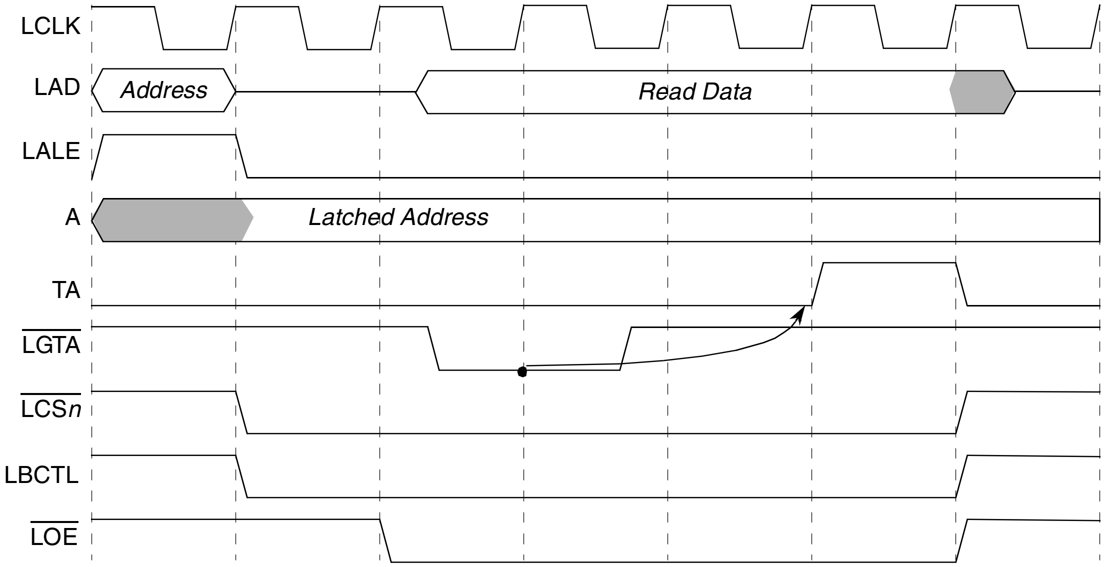


**Figure 13-45. External Termination of GPCM Access**

### 13.4.2.5 GPCM Boot Chip-Select Operation

Boot chip-select operation allows address decoding for a boot ROM before system initialization. $\overline{\text{LCS0}}$ is the boot chip-select output; its operation differs from other external chip-select outputs after a system reset. When the core begins accessing memory after system reset, $\overline{\text{LCS0}}$ is asserted for every local bus access until BR0 or OR0 is reconfigured.

The boot chip-select also provides a programmable port size, which is configured during reset. The boot chip-select does not provide write protection. $\overline{\text{LCS0}}$ operates this way until the first write to OR0 and it can be used as any other chip-select register after the preferred address range is loaded into BR0. After the first write to OR0, the boot chip-select can be restarted only with a hardware reset. Table 13-34 describes the initial values of the boot bank in the memory controller.

Table 13-34. Boot Bank Field Values after Reset for GPCM as Boot Controller

| Register | Field | Setting |
|---|---|---|
| BR0 | BA | 0000_0000_0000_0000_0 |
| | PS | *From cfg_rom_loc* |
| | DECC | 00 |
| | WP | 0 |
| | MSEL | 000 |
| | ATOM | 00 |
| | V | 1 |
| OR0 | AM | 0000_0000_0000_0000_0 |
| | BCTLD | 0 |
| | CSNT | 1 |
| | ACS | 11 |
| | XACS | 1 |
| | SCY | 1111 |
| | SETA | 0 |
| | TRLX | 1 |
| | EHTR | 1 |
| | EAD | 1 |

## 13.4.3 Flash Control Machine (FCM)

The FCM provides a glueless interface to parallel-bus NAND Flash EEPROM devices. The FCM contains three basic configuration register groups—BR*n*, OR*n*, and FMR.

Figure 13-46 shows a simple connection between an 8-bit port size NAND Flash EEPROM and the eLBC in FCM mode. Commands, address bytes, and data are all transferred on LAD[0:7][1], with $\overline{\text{LFWE}}$ asserted for transfers written to the device, or $\overline{\text{LFRE}}$ asserted for transfers read from the device. eLBC signals LFCLE and LFALE determine whether writes are of type command (only LFCLE asserted), address (only LFALE asserted), or write data (neither LFCLE nor LFALE asserted). The NAND Flash RDY/$\overline{\text{BSY}}$ pin is normally open-drain, and should be pulled high by a 4.7-KΩ resistor. On system reset, a global (boot) chip-select is available that provides a boot ROM chip-select ($\overline{\text{LCS0}}$) prior to the system being fully configured.

[1] Note bit numbering reversal: LAD[0] (msb) connects to Flash IO[7], while LAD[7] (lsb) connects to IO[0].

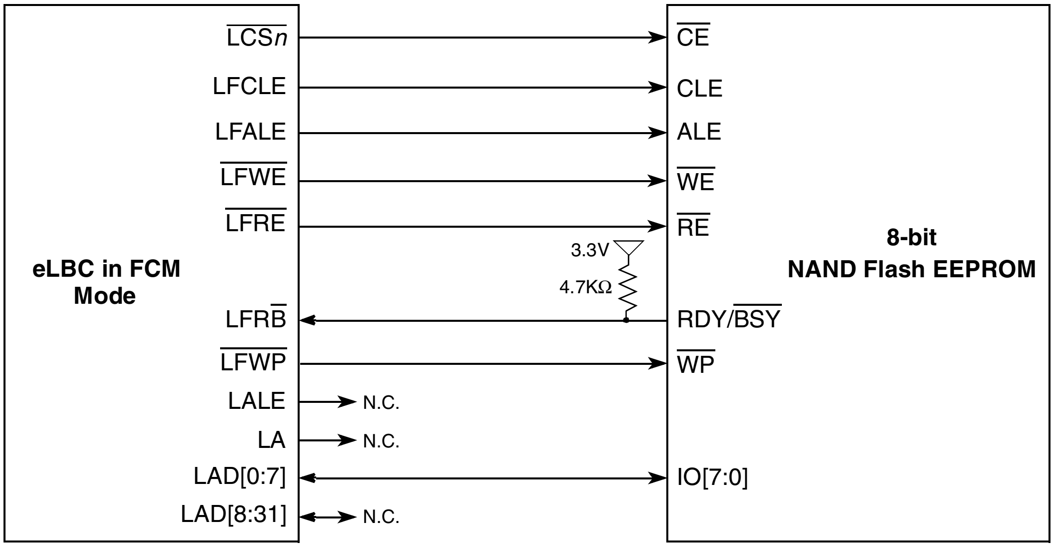


**Figure 13-46. Local Bus to 8-Bit FCM Device Interface**

Basic read access timing for FCM is shown in Figure 13-47. Although LCLK is shown for reference, NAND Flash EEPROMs do not make use of the clock.

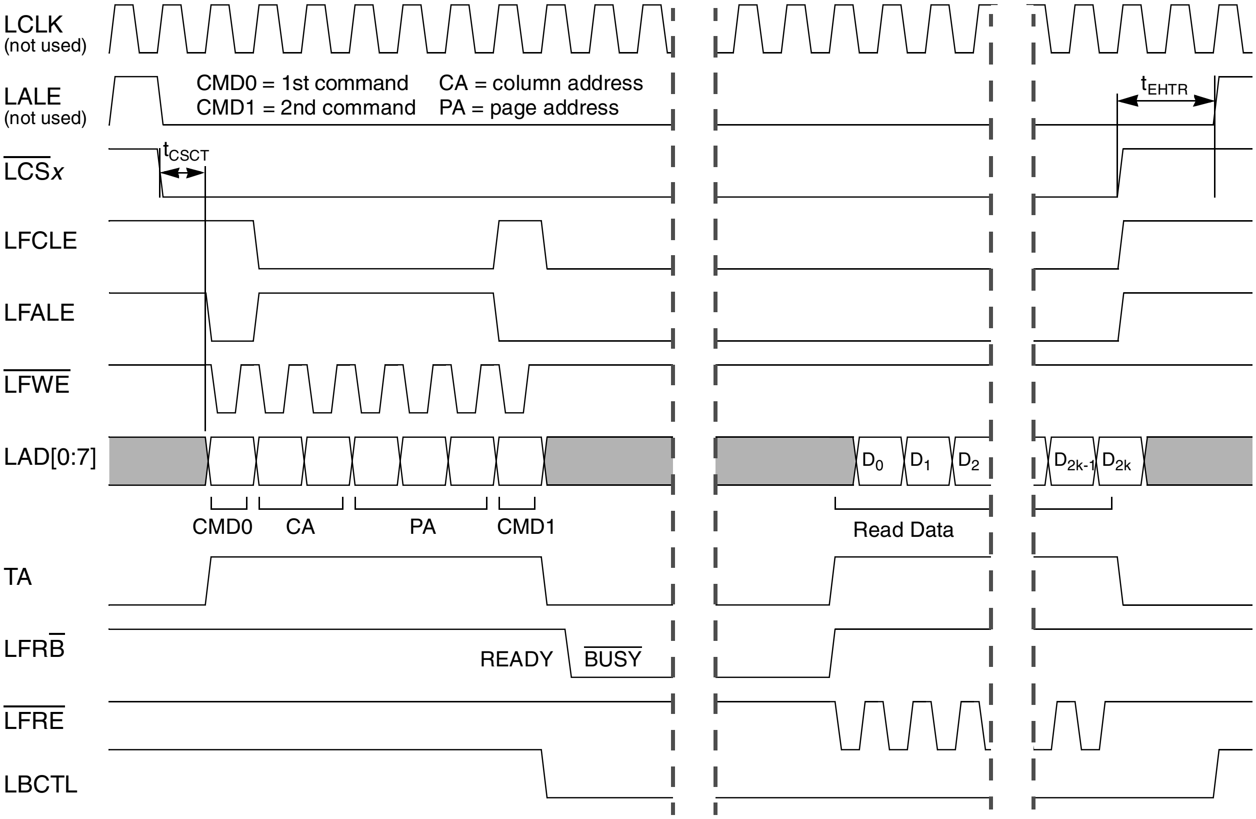


**Figure 13-47. FCM Basic Page Read Timing**
**(PGS = 1, CSCT = 0, CST = 0, CHT = 1, RST = 1, SCY = 0, TRLX = 0, EHTR = 1)**

Following the assertion of LALE, FCM asserts $\overline{\mathrm{LCS}n}$ to commence a command sequence to the Flash device. After a delay of $t_{CSCT}$, the first command can be written to the device on assertion of $\overline{\mathrm{LFWE}}$, followed by any parameters (typically address bytes and data), and concluded with a secondary command. In many cases, the second command initiates a long-running operation inside the Flash device, which pulls the wired-OR pin LFR$\overline{\mathrm{B}}$ low to indicate that the device is busy. Since in Figure 13-47 FCM is now expecting a read response, it takes LBCTL low to turnaround any bus transceivers that are present. Upon LFR$\overline{\mathrm{B}}$ indicating ready status, FCM asserts $\overline{\mathrm{LFRE}}$ repeatedly to recover bytes of read data, and the bytes are stored in eLBC's FCM buffer RAM while an ECC is optionally computed on the bytes transferred. Finally, FCM negates $\overline{\mathrm{LCS}n}$ and delays eLBC by $t_{EHTR}$ before any subsequent memory access occurs.

### 13.4.3.1 FCM Buffer RAM

Read and write accesses to eLBC banks controlled by FCM do not access attached NAND Flash EEPROMs directly. Rather, these accesses read and write the FCM buffer RAM—a single, shared 8-Kbyte space internal to eLBC and mapped by the base address of every FCM bank. Even though each FCM-controlled bank will have a different base address to differentiate it, all accesses to such banks will access the same buffer space. External eLBC signals, such as LALE and $\overline{\mathrm{LCS}n}$, will not assert upon accesses to the buffer RAM. The FCM buffer RAM is logically divided into two or more buffers, depending on the setting of OR*n*[PGS], with different buffers being accessible concurrently by software and FCM.

To perform a page read operation from a NAND Flash device, software initializes the FCM command, mode, and address registers, before issuing a special operation (FMR[OP] set non-zero) to a particular FCM-controlled bank. FCM will execute the sequence of op-codes held in FIR, reading data from the Flash device into the shared buffer RAM. While this read is taking place, software is free to access any data stored in other, currently inactive buffers of the FCM buffer RAM through reads or writes to any bank controlled by FCM. If command completion interrupts are enabled, an interrupt will be generated once FCM has completed the read. When FCM has completed its last command, software can switch to the newly read buffer and issue further commands.

To perform a page write operation, software first prepares data to be written in a fresh buffer. Then, the FCM command, mode, and address registers are initialized, and a special operation (FMR[OP] set non-zero) is issued to a particular FCM-controlled bank. FCM will execute the sequence of op-codes held in FIR, writing data from shared buffer RAM to the Flash device. To ensure that the device is enabled for programming, software must initialize FMR[OP] = 11, which prevents assertion of $\overline{\mathrm{LFWP}}$ during the write. While this write is taking place, software is free to access any data stored in other, currently inactive buffers of the FCM buffer RAM through reads or writes to any bank controlled by FCM. When FCM has completed its last command, software can re-use the previously written buffer and issue further commands.

See Section 13.4.3.4.2, “Boot Block Loading into the FCM Buffer RAM,” for a description of the shared buffer RAM layout during boot.

#### 13.4.3.1.1 Buffer Layout and Page Mapping for Small-Page NAND Flash Devices

The FCM buffer space is divided into eight 1-Kbyte buffers for small-page devices (OR*n*[PGS] = 0), mapped as shown in Figure 13-48. Each page in a small-page NAND Flash comprises 528 bytes, where

512 bytes appear as main region data, and 16 bytes appear as spare region data. The EEPROM's page numbered *P* is associated with buffer number (*P* mod 8), where *P* = FPAR[PI]. Since the bank size set by OR*n*[AM] will be greater than 8 Kbytes, an identical image of the FCM buffer RAM appears replicated every 8 Kbytes throughout the bank address space. It is recommended that the bank size be set to 32 Kbytes, which covers a single NAND Flash block for small-page devices.

For FCM commands, register FPAR sets the page address and, therefore, also the buffer number.In the case that FBCR[BC] = 0, FCM transfers an entire page, comprising the 512-byte main region followed by the 16-byte spare region; the 496-byte reserved region is not accessed, and remains undefined for software. However, for commands given a specific byte count in FBCR[BC], FPAR[MS] locates the starting address in either the main region (MS = 0) or the spare region (MS = 1). Where different eLBC banks control both small and large-page devices, a large-page 4-Kbyte buffer must be assigned to either the first 4 or last 4 small-page buffers.

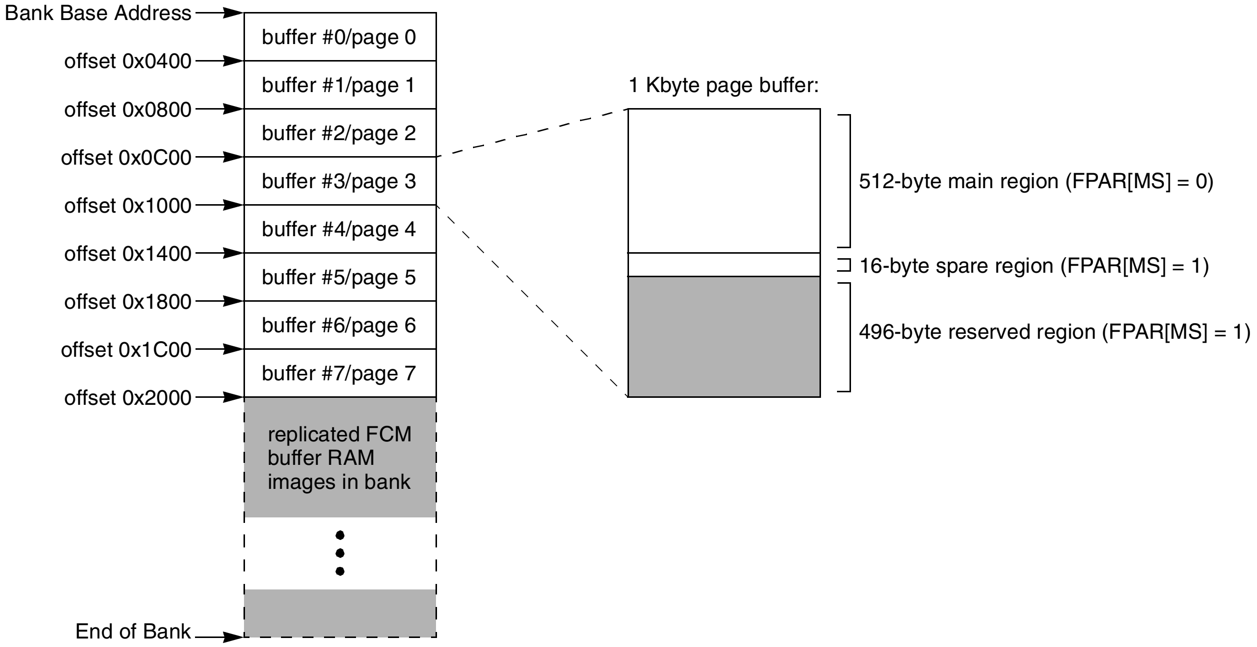


**Figure 13-48. FCM Buffer RAM Memory Map for Small-Page (512-byte page) NAND Flash Devices**

### 13.4.3.1.2 Buffer Layout and Page Mapping for Large-Page NAND Flash Devices

The FCM buffer space is divided into two 4 Kbyte buffers for large-page devices (OR*n*[PGS] = 1), mapped as shown in Figure 13-49. Each page in a large-page NAND Flash comprises 2112 bytes, where 2048 bytes appear as main region data, and 64 bytes appear as spare region data. The EEPROM's page numbered *P* is associated with buffer number (*P* mod 2), where *P* = FPAR[PI]. Since the bank size set by OR*n*[AM] will be greater than 8 Kbytes, an identical image of the FCM buffer RAM appears replicated every 8 Kbytes throughout the bank address space. It is recommended that the bank size be set to 256 Kbytes, which covers a single NAND Flash block for large-page devices.

For FCM commands, register FPAR sets the page address and, therefore, also the buffer number. In the case that FBCR[BC] = 0, FCM transfers an entire page, comprising the 2048-byte main region followed by the 64-byte spare region; the 1984-byte reserved region is not accessed, and remains undefined for software. However, for commands given a specific byte count in FBCR[BC], FPAR[MS] locates the starting address in either the main region (MS = 0) or the spare region (MS = 1). Where different eLBC banks control both small and large-page devices, a large-page 4 Kbyte buffer must be assigned to either the first 4 or last 4 small-page buffers.

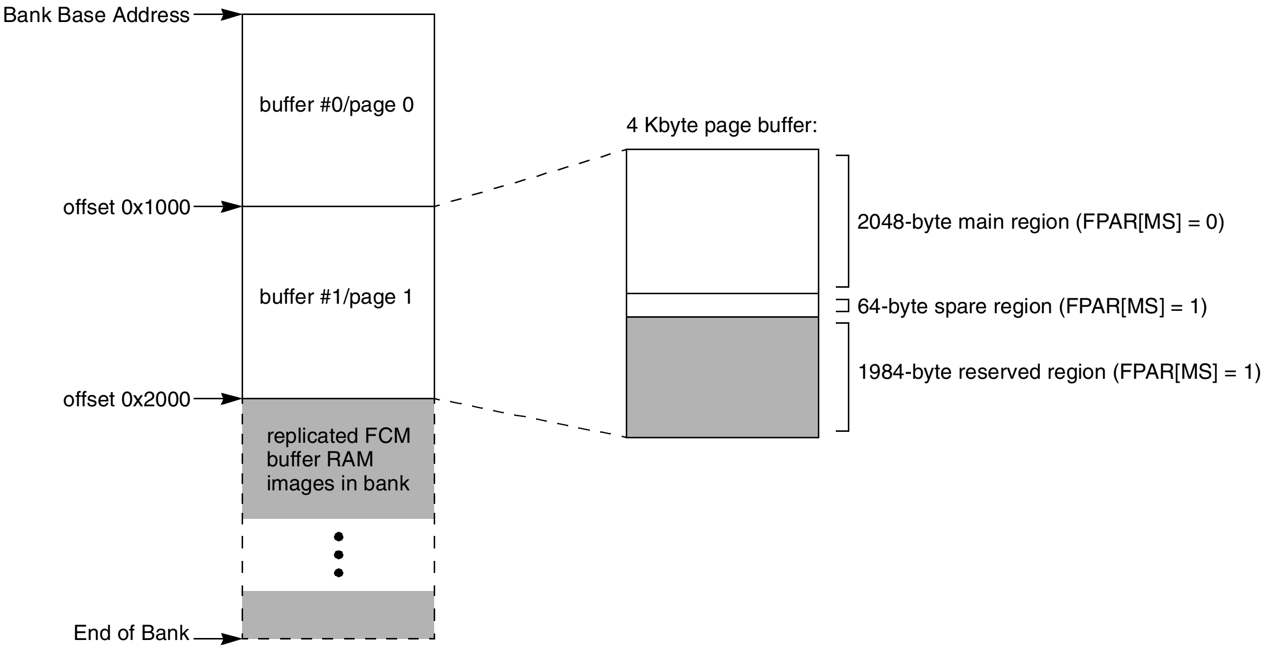

Bank Base Address
buffer #0/page 0
offset 0x1000
buffer #1/page 1
offset 0x2000
replicated FCM buffer RAM images in bank
End of Bank
4 Kbyte page buffer:
2048-byte main region (FPAR[MS] = 0)
64-byte spare region (FPAR[MS] = 1)
1984-byte reserved region (FPAR[MS] = 1)

**Figure 13-49. FCM Buffer RAM Memory Map for Large-Page (2-Kbyte page) NAND Flash Devices**

### 13.4.3.1.3 Error Correcting Codes and the Spare Region

The FCM's ECC engine makes use of data in the NAND Flash spare region to store pre-computed ECC code words. ECC is calculated in a single pass over blocks of 512 bytes of data in the main region. The setting of FMR[ECCM] determines the location of the 24-bit ECC in the spare region.

The basic ECC algorithm is depicted in Figure 13-50. The stream of data bytes is considered to form a matrix having 8 columns (corresponding with the device bus IO[7:0] or IO[15:8]) and 512 rows (corresponding with each byte in the ECC block). Six bits of parity, {$P_4$, $P_4$', $P_2$, $P_2$', $P_1$, $P_1$'}, are calculated across the columns, and at most 18 bits of parity {$P_{2048}$, $P_{2048}$', ..., $P_{16}$, $P_{16}$', $P_8$, $P_8$'} are calculated across the rows to create a 24-bit Hamming code for the data block. In this calculation, parity bit $P_N$' is the exclusive-OR of every alternate *N*-bit group of bits positioned at even intervals (starting at *N*-bit group 0, then continuing to group 2, 4, etc.), while parity bit $P_N$ is the exclusive-OR of every alternate *N*-bit group of bits positioned at odd intervals (starting at *N*-bit group 1, then continuing to group 3, 5, etc.).

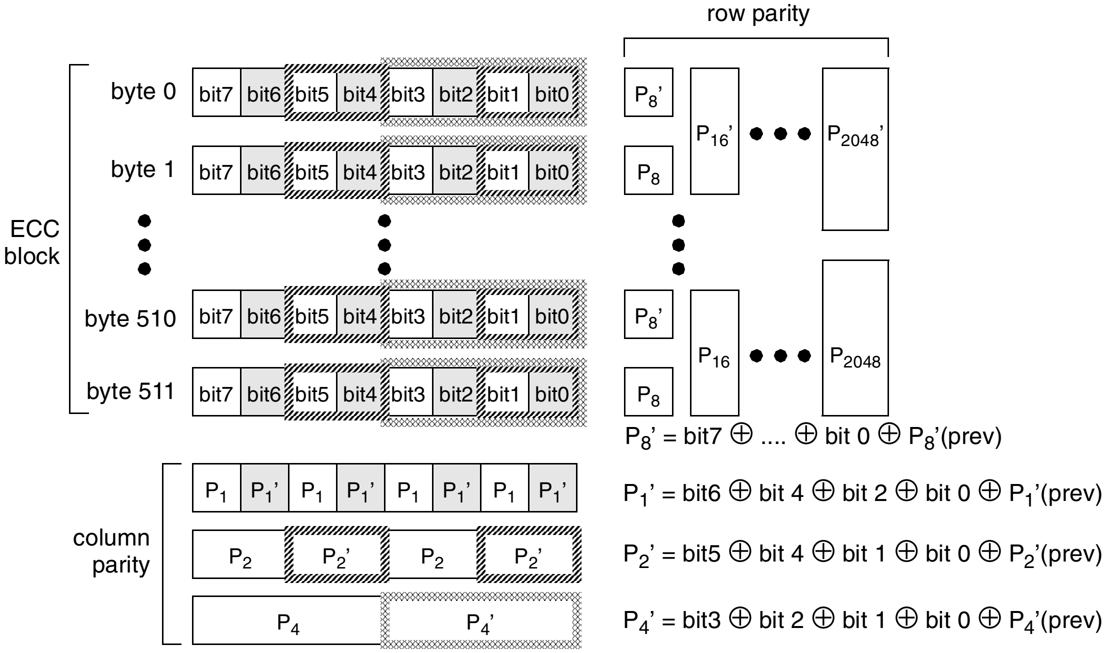

row parity
ECC block
byte 0
byte 1
byte 510
byte 511
bit7 bit6 bit5 bit4 bit3 bit2 bit1 bit0
P8'
P8
P16'
P16
P2048'
P2048
P8' = bit7 ⊕ .... ⊕ bit 0 ⊕ P8'(prev)
column parity
P1 P1' P1 P1' P1 P1' P1 P1'
P2 P2' P2 P2'
P4 P4'
P1' = bit6 ⊕ bit 4 ⊕ bit 2 ⊕ bit 0 ⊕ P1'(prev)
P2' = bit5 ⊕ bit 4 ⊕ bit 1 ⊕ bit 0 ⊕ P2'(prev)
P4' = bit3 ⊕ bit 2 ⊕ bit 1 ⊕ bit 0 ⊕ P4'(prev)

**Figure 13-50. FCM ECC Calculation**

The 24-bit ECC code word format is shown in Figure 13-51 for normal ECC polarity. Setting LBCR[EPAR] = 1 changes ECC polarity, and thus omits negation of each $P_N$ and $P_N$’ bit.

|  | 0 (MSB) | 1 | 2 | 3 | 4 | 5 | 6 | 7 (LSB) |
|---|---|---|---|---|---|---|---|---|
| EC0 | $\sim P_{64}$ | $\sim P_{64}'$ | $\sim P_{32}$ | $\sim P_{32}'$ | $\sim P_{16}$ | $\sim P_{16}'$ | $\sim P_{8}$ | $\sim P_{8}'$ |
| EC1 | $\sim P_{1024}$ | $\sim P_{1024}'$ | $\sim P_{512}$ | $\sim P_{512}'$ | $\sim P_{256}$ | $\sim P_{256}'$ | $\sim P_{128}$ | $\sim P_{128}'$ |
| EC2 | $\sim P_{4}$ | $\sim P_{4}'$ | $\sim P_{2}$ | $\sim P_{2}'$ | $\sim P_{1}$ | $\sim P_{1}'$ | $\sim P_{2048}$ | $\sim P_{2048}'$ |

**Figure 13-51. ECC Layout for LBCR[EPAR] = 0 (~ represents logical negation)**

The placement of ECC code words in relation to FMR[ECCM] is shown in Figure 13-52. For small-page devices, only a single 512-byte main region is ECC-protected. For large-page devices, there are four adjacent main regions, and each has a 16-byte spare region—of which only one is shown in the figure. If eLBC is configured to generate ECC (BR*n*[DECC] = 10), FCM will substitute on full-page write transfers the three code word bytes in place of the spare region data originally provided at the locations shown in Figure 13-52 and write the same 24-bit ECC code in the appropriate FECC*n* register for software reference. Transfers shorter than a full page, however, require software to prepare the appropriate ECC in the spare region. Similarly, FCM can check and correct bit errors on full-page reads if BR*n*[DECC] = 01 or 10. A correctable error is a single bit error in any 512-byte block of main region data, as judged by comparison of a regenerated ECC with the ECC retrieved from the spare region, or a single bit error in the retrieved ECC only. Bit errors in the main region are corrected before FCM completes its final read transfer and signals an event in LTESR[CC]. The bit vector in LTECCR[SBCE] can be checked on FCM CC event to find out if any 512-byte block or the corresponding ECC have single bit correctable errors. Errors that appear more complex (two or more bits in error per 512-byte block) are not corrected, but are flagged as parity errors by FCM. The bit vector in LTEATR[PB] or LTECCR[MBUE] can be checked to determine which 512-byte blocks in a large-page NAND Flash main region were found to be uncorrectable.

| ECCM | Byte 0 | Byte 511 | Other Mains | Spare 0 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Main Region | | | — | | EC0 | EC1 | EC2 | — | | | | | | |
| 1 | Main Region | | | — | | | | EC0 | EC1 | EC2 | — | | | | |

Figure 13-52. ECC Placement in NAND Flash Spare Regions in Relation to FMR[ECCM]

### 13.4.3.2 Programming FCM

FCM has a fully general command and data transfer sequencer that caters for both common and specific/proprietary NAND Flash command sequences. The command sequencer reads a program out of the FIR register, which can hold up to 8 instructions, each represented by a 4-bit op-code, as illustrated in Figure 13-53. The first instruction executed is read from FIR[OP0], the next is read from FIR[OP1], and likewise to subsequent instructions, ending at FIR[OP7] or until the only instructions remaining are NOPs. If FIR contains nothing but NOP instructions, FCM will not assert $\overline{\text{LCS}n}$, otherwise, $\overline{\text{LCS}n}$ is asserted prior to the first instruction and remains asserted until the last instruction has completed. If LTESR[CC] is enabled, completion of the last instruction will trigger a command completion event interrupt from eLBC.

Prior to executing a sequence, necessary operands for the instructions will need to be set in the FMR, FCR, MDR, FBCR, FBAR, and FPAR registers. The AS0–AS3 address and data pointers associated with FCM's use of MDR all reset to select AS0 at the start of the instruction sequence. A complete list of op-codes can be found in Section 13.3.1.18, "Flash Instruction Register (FIR)."

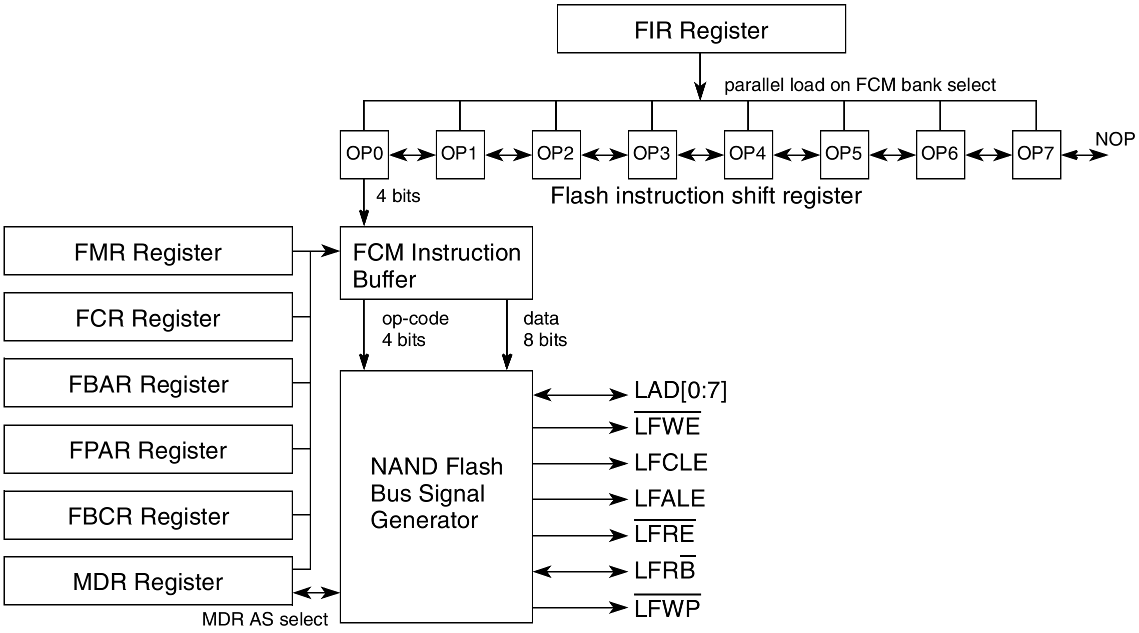


Figure 13-53. FCM Instruction Sequencer Mechanism

#### 13.4.3.2.1 FCM Command Instructions

There are two kinds of command instruction:

- Commands that issue immediately—CM0, CM1, CM2, and CM3. These commands write a single command byte by asserting LFCLE and $\overline{\text{LFWE}}$ while driving an 8-bit command onto LAD[0:7]. Op-code CM*n* sources its command byte from field FCR[CMD*n*], therefore up to four different commands can be issued in any FCM instruction sequence.
- Commands that wait for LFR$\overline{\text{B}}$ to be sampled high (EEPROM in ready state) before issuing—CW0, and CW1. These commands first poll the LFR$\overline{\text{B}}$ pin, waiting for it to go high, before writing a single command byte onto LAD[0:7], sourced from FCR[CMD*n*] for op-code CW*n*. It is necessary to use CW*n* op-codes whenever the EEPROM is expected to be in a busy state (such as following a page read, block erase, or program operation) and therefore initially unresponsive to commands. To avoid deadlock in cases where the device is already available, FCM does not expect a transition on LFR$\overline{\text{B}}$. Rather, FCM waits for 8×(2+OR*n*[SCY]) clock cycles (when OR*n*[TRLX] = 0) or 16×(2+OR*n*[SCY]) clock cycles (when OR*n*[TRLX] = 1) before sampling the level of LFR$\overline{\text{B}}$. If the level of LFR$\overline{\text{B}}$ does not return high before a time-out set by FMR[CWTO] occurs, FCM proceeds to issue the command normally, and a FCT event is issued to LTESR.

The manufacturer's datasheet should be consulted to determine values for programming into the FCR register, and whether a given command in the sequence is expected to initiate busy device behavior.

#### 13.4.3.2.2 FCM No-Operation Instruction

A NOP instruction that appears in FIR ahead of the last instruction is executed with the timing of a regular command instruction, but neither LFCLE nor $\overline{\text{LFWE}}$ are asserted. Thus a NOP instruction may be used to insert a pause matching the time taken for a regular command write.

#### 13.4.3.2.3 FCM Address Instructions

Address instructions are used to issue addresses to the NAND Flash EEPROM. A complete device address is formed from a sequence of one or more bytes, each written onto LAD[0:7] with LFALE and $\overline{\text{LFWE}}$ asserted together. There are three kinds of address generation provided:

- Column address—CA. A column address comprises one byte (OR*n*[PGS] = 0) or two bytes (OR*n*[PGS] = 1) locating the starting byte or word to be transferred in the next page read or write sequence. FPAR[CI] sets the value of the column index provided that FBCR[BC] is non-zero. In the case that FBCR[BC] = 0, a column index of zero is issued to the device, regardless of the value in FPAR[CI].
- Page address—PA. A page address comprises 2, 3, or 4 bytes, depending on the setting of FMR[AL], and locates the data page in the NAND Flash address space. The complete page address is the concatenation of the block index, read from FBAR[BLK], with the page-in-block index, read from FPAR[PI]. The page address length set in FMR[AL] should correspond with the size of EEPROM being accessed. Similarly, the block index in FBAR[BLK] must not exceed the maximum block index for the device, as most devices require reserved address bits to be written as zero.

- User-defined address—UA. This instruction allows the FCM to write a user-defined address byte, which is read from the next AS field in MDR, starting at MDR[AS0]. Each subsequent UA instruction reads an adjacent AS field in MDR, until all four AS bytes (MDR[AS0], MDR[AS1], MDR[AS2], MDR[AS3]) have been sent; a fifth and any following UA instructions send zero as the address byte. Note that each UA instruction advances the MDR pointer for writes by one byte, and therefore a mix of UA and WS instructions can consume adjacent bytes from MDR.

#### 13.4.3.2.4 FCM Data Read Instructions

Data read instructions assert $\overline{\mathrm{LFRE}}$ repeatedly to transfer one or more bytes of read data from the NAND Flash EEPROM. Data read instructions are distinguished by their data destination:

- Read data to buffer RAM immediately—RB. This instruction reads FBCR[BC] bytes of data into the current FCM RAM buffer addressed by FPAR. If FBCR[BC] = 0, an entire page (including spare region) is transferred in a burst, starting at the page boundary, and the ECC calculation is checked against the ECC stored in the spare region. Correctable ECC errors are corrected and reported in LTECCR[SBCE]; other errors may cause an interrupt if enabled. If the value of FBCR[BC] takes the read pointer beyond the end of the spare region in the buffer, FCM discards any excess bytes read.
- Read data/status to MDR immediately—RS. This instruction asserts $\overline{\mathrm{LFRE}}$ exactly once to read one byte (8-bit port size) of data into the next AS field of MDR. Reads beyond the fourth byte of MDR are discarded. The MDR read pointer is independent of the MDR write pointer used by UA and WS instructions.
- Read data to buffer RAM once waited on ready—RBW. This instruction first polls the LFR$\overline{\mathrm{B}}$ pin, waiting for it to go high, before proceeding with a read to buffer as described for the RB instruction. Sampling and time-outs for polling the LFR$\overline{\mathrm{B}}$ pin follow the behavior of CW*n* instructions.
- Read data/status to MDR once waited on ready—RSW. This instruction first polls the LFR$\overline{\mathrm{B}}$ pin, waiting for it to go high, before proceeding with a status read to MDR as described for the RS instruction. Sampling and time-outs for polling the LFR$\overline{\mathrm{B}}$ pin follow the behavior of CW*n* instructions.

#### 13.4.3.2.5 FCM Data Write Instructions

Data write instructions assert $\overline{\mathrm{LFWE}}$ repeatedly (with LFCLE and LFALE both negated) to transfer one or more bytes of write data to the NAND Flash EEPROM. Data write instructions are distinguished by their data source:

- Write data from FCM buffer RAM—WB. This instruction writes FBCR[BC] bytes of data from the current FCM RAM buffer addressed by FPAR. If FBCR[BC] = 0, an entire page (including spare region) is transferred in a burst, starting at the page boundary, and the ECC calculation is stored in the appropriate FECC*n* registers and spare region in accordance with the setting of FMR[ECCM]. If the value of FBCR[BC] takes the write pointer beyond the end of the spare region in the buffer, the value of data written by FCM is undefined.
- Write data/status from MDR—WS. This instruction asserts $\overline{\mathrm{LFWE}}$ exactly once to write one byte (8-bit port size) of data taken from the next AS field of MDR. Attempts to write beyond four bytes of MDR has the effect of writing zeros. The MDR write pointer is independent of the MDR read pointer used by RS and RSW instructions.

### 13.4.3.3 FCM Signal Timing

If BR*n*[MSEL] selects the FCM, the attributes for the memory cycle are taken from OR*n*. These attributes include the CSCT, CST, CHT, RST, SCY, TRLX, and EHTR fields.

#### 13.4.3.3.1 FCM Chip-Select Timing

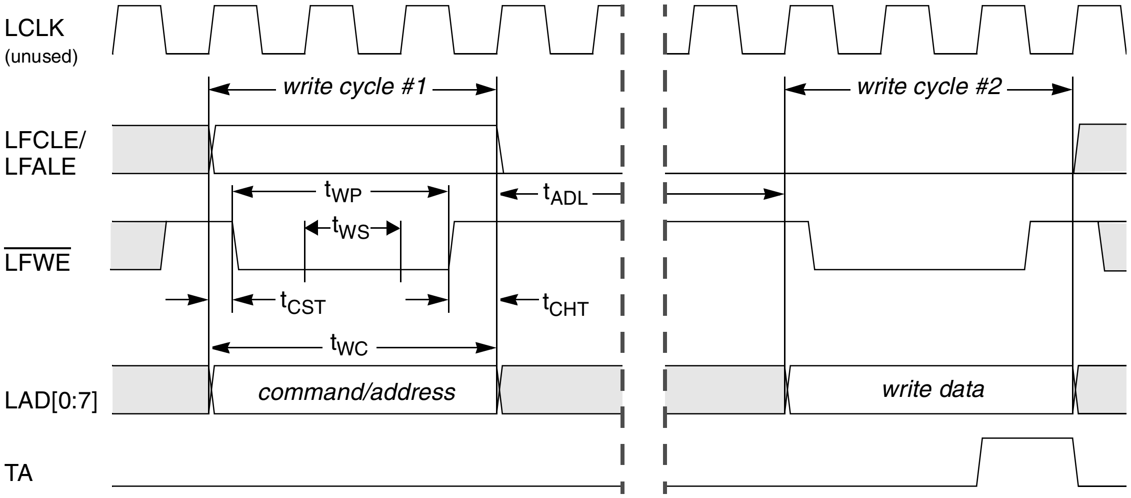
The timing of $\overline{\text{LCS}n}$ assertion in FCM mode is illustrated by the timing diagram in Figure 13-47. $\overline{\text{LCS}n}$ is asserted immediately following LALE negation, and remains asserted until the last instruction in FIR has completed. The delay, $t_{CSCT}$, between $\overline{\text{LCS}n}$ assertion and commencement of the first NAND Flash instruction is controlled by OR*n*[CSCT] and OR*n*[TRLX], as shown in Table 13-35. OR*n*[CSCT] should be set in accordance with the NAND Flash EEPROM chip-select to $\overline{\text{WE}}$ set-up time specification.

**Table 13-35. FCM Chip-Select to First Command Timing**

| ORn[TRLX] | ORn[CSCT] | $\overline{\text{LCS}n}$ to First Command Delay |
|---|---|---|
| 0 | 0 | 1 LCLK clock cycle |
| 0 | 1 | 4 LCLK clock cycles |
| 1 | 0 | 2 LCLK clock cycles |
| 1 | 1 | 8 LCLK clock cycles |

#### 13.4.3.3.2 FCM Command, Address, and Write Data Timing

The FCM command (CM0–CM3, CW0, CW1), address (CA, PA, UA), and data write (WB, WS) instructions all share the same basic timing attributes. Assertion of $\overline{\text{LFWE}}$ initiates transfer via LAD[0:7], and the options in OR*n* for FCM mode establish the set-up, hold, and wait state timings with respect to $\overline{\text{LFWE}}$, as shown in Figure 13-54.



**Notes:** $t_{CST}$ = Command to $\overline{\text{LFWE}}$ set-up time.
$t_{CHT}$ = Command to $\overline{\text{LFWE}}$ hold time.
$t_{ADL}$ = Command/address to write data delay.
$t_{WP}$ = $\overline{\text{LFWE}}$ pulse time, driven low.
$t_{WS}$ = Command wait state time.
$t_{WC}$ = Command cycle time.

**Figure 13-54. Timing of FCM Command/Address and Write Data Cycles (for TRLX = 0, CHT = 0, CST = 1, SCY = 1)**

The timing parameters are summarized in Table 13-36.

**Table 13-36. FCM Command, Address, and Write Data Timing Parameters**

| Option Register Attributes | | | Timing Parameter (LCLK Clock Cycles)[1] | | | | | |
|---|---|---|---|---|---|---|---|---|
| **TRLX** | **CHT** | **CST** | $t_{CST}$ | $t_{CHT}$ | $t_{WS}$ | $t_{WP}$ | $t_{WC}$ | $t_{ADL}$ |
| 0 | 0 | 0 | 0 | ½ | SCY | 1½+SCY | 2+SCY | 4×(2+SCY) |
| 0 | 0 | 1 | ¼ | ½ | SCY | 1¼+SCY | 2+SCY | 4×(2+SCY) |
| 0 | 1 | 0 | 0 | 1 | SCY | 1+SCY | 2+SCY | 4×(2+SCY) |
| 0 | 1 | 1 | ¼ | 1 | SCY | ¾+SCY | 2+SCY | 4×(2+SCY) |
| 1 | 0 | 0 | ½ | 1½ | 2×SCY | 1+2×SCY | 3+2×SCY | 8×(2+SCY) |
| 1 | 0 | 1 | 1 | 1½ | 2×SCY | ½+2×SCY | 3+2×SCY | 8×(2+SCY) |
| 1 | 1 | 0 | ½ | 2 | 2×SCY | ½+2×SCY | 3+2×SCY | 8×(2+SCY) |
| 1 | 1 | 1 | 1 | 2 | 2×SCY | 2×SCY | 3+2×SCY | 8×(2+SCY) |

[1] In the parameters, SCY refers to a delay of OR*n*[SCY] clock cycles.

An example of minimum delay command timing appears in Figure 13-55. Note that the set-up, wait-state, and hold timing of command, address, and write data cycles with respect to $\overline{\text{LFWE}}$ assertion are all identical, and that the minimum cycle extends for two LCLK clock cycles.

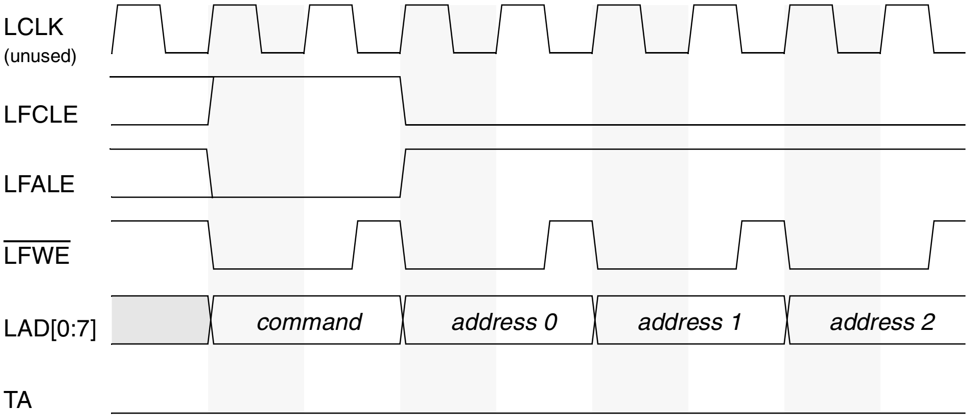


**Figure 13-55. Example of FCM Command and Address Timing with Minimum Delay Parameters (for TRLX = 0, CHT = 0, CST = 0, SCY = 0)**

An example of relaxed command timing is shown in Figure 13-56.

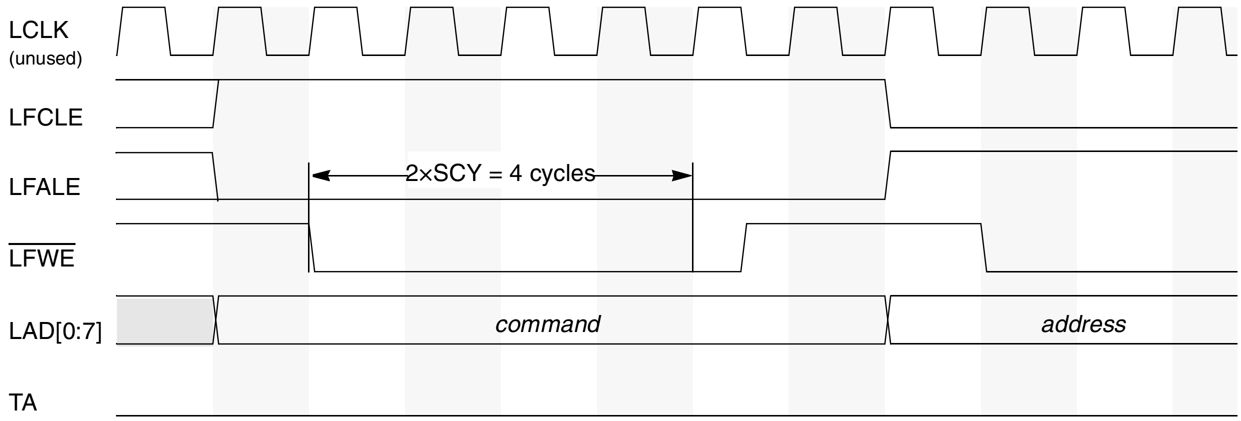


**Figure 13-56. Example of FCM Command and Address Timing with Relaxed Parameters (for TRLX = 1, CHT = 0, CST = 1, SCY = 2)**

#### 13.4.3.3.3 FCM Ready/Busy Timing

Instructions CW0, CW1, RBW, and RSW force FCM to observe the state of the LFR$\overline{\text{B}}$ pin, which may be driven low by a long-latency NAND Flash operation, such as a page read. Following the issue of such commands, FCM waits as shown in Figure 13-57 before sampling the state of LFR$\overline{\text{B}}$. This guards against observing LFR$\overline{\text{B}}$ before it has been properly driven low by the device, but does not preclude LFR$\overline{\text{B}}$ from remaining high after a command. In addition, FCM samples and compares the state of LFR$\overline{\text{B}}$ on two consecutive cycles of LCLK to filter out noise on this signal as it rises to the ready state (LFR$\overline{\text{B}}$ = 1).

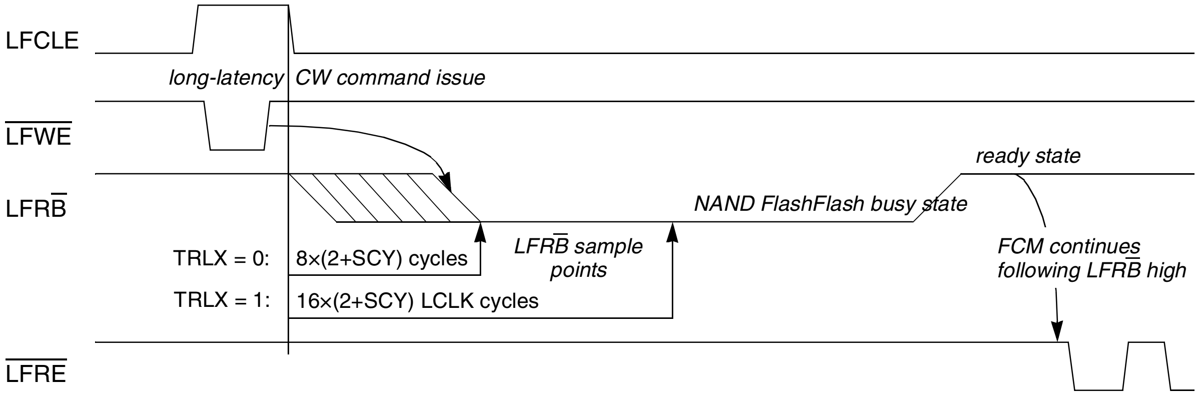
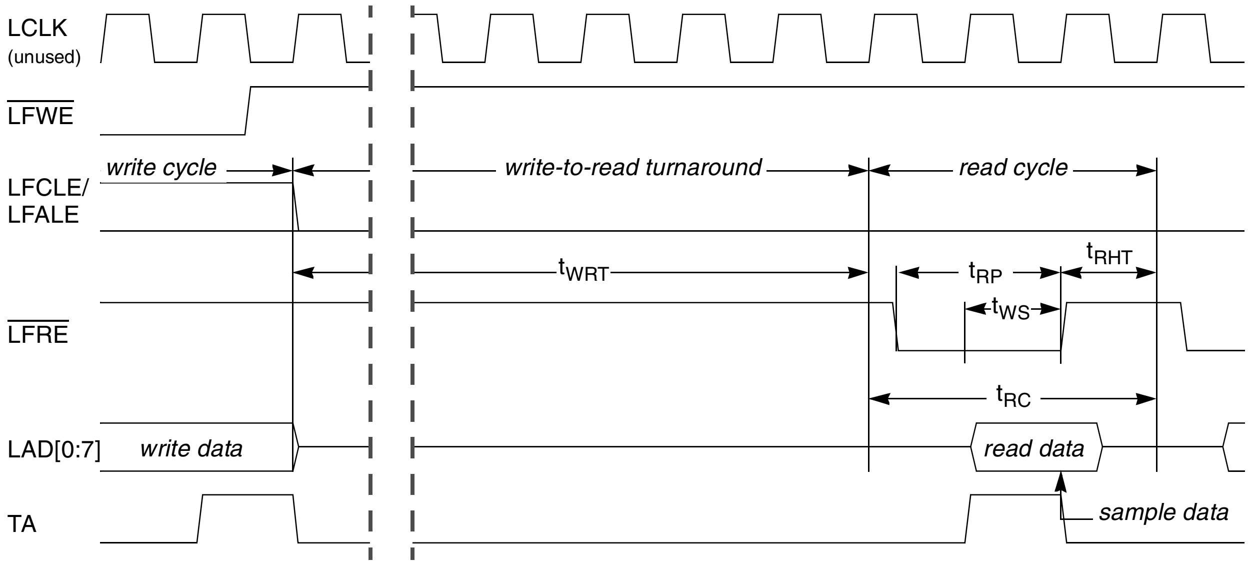


**Figure 13-57. FCM Delay Prior to Sampling LFR$\overline{\text{B}}$ State**

#### 13.4.3.3.4 FCM Read Data Timing

The timing for read data transfers is shown in Figure 13-58. Upon assertion of $\overline{\text{LFRE}}$, the Flash device will enable its output drivers and drive valid read data while $\overline{\text{LFRE}}$ is held low. FCM samples read data on the rising edge of $\overline{\text{LFRE}}$, which follows an optional number of wait states. Note that FCM will delay the first read if a RBW or RSW instruction is issued, in which case LFR$\overline{\text{B}}$ sample timing takes effect (see Section 13.4.3.3.3, “FCM Ready/Busy Timing”).



**Notes:** $t_{RP}$ = $\overline{\text{LFRE}}$ pulse time, read period.
$t_{RHT}$ = $\overline{\text{LFRE}}$ hold time.
$t_{WRT}$ = Write to read turnaround time.
$t_{WS}$ = Read wait state time.
$t_{RC}$ = Read data cycle time.

**Figure 13-58. FCM Read Data Timing (for TRLX = 0, RST = 0, SCY = 1)**

The timing parameters are summarized in Table 13-37.

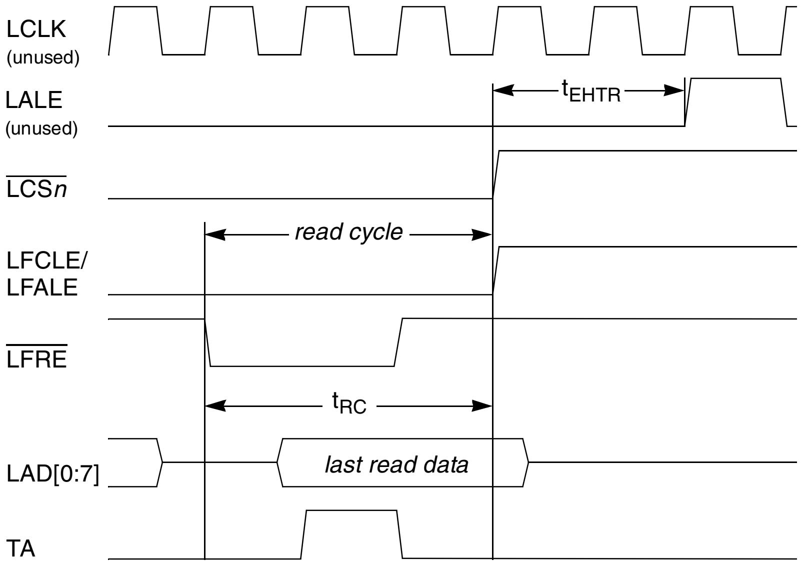
**Table 13-37. FCM Read Data Timing Parameters**

| Option Register Attributes | | Timing Parameter (LCLK Clock Cycles)[1] | | | | |
|---|---|---|---|---|---|---|
| TRLX | RST | $t_{RP}$ | $t_{RHT}$ | $t_{WS}$ | $t_{RC}$ | $t_{WRT}$ |
| 0 | 0 | ¾+SCY | 1 | SCY | 2+SCY | 4×(2+SCY) |
| 0 | 1 | 1+SCY | 1 | SCY | 2+SCY | 4×(2+SCY) |
| 1 | 0 | ½+2×SCY | 2 | 2×SCY | 3+2×SCY | 8×(2+SCY) |
| 1 | 1 | 1+2×SCY | 2 | 2×SCY | 3+2×SCY | 8×(2+SCY) |

[1] In the parameters, SCY refers to a delay of OR*n*[SCY] clock cycles.

### 13.4.3.3.5 FCM Extended Read Hold Timing

Allowance for slow output driver turn-off when reading NAND Flash EEPROMs is made via setting of OR*n*[EHTR] and OR*n*[TRLX]. The extended read data hold time, shown at $t_{EHTR}$ in Figure 13-47 and Figure 13-59, is a delay inserted by FCM between the last data read and another eLBC memory access (requiring LALE assertion). $\overline{\text{LCS}n}$ is negated during $t_{EHTR}$ to allow external devices and bus transceivers time to disable their drivers.



**Notes:** $t_{RC}$ = Read data cycle time.
$t_{EHTR}$ = Extended read data hold time.

**Figure 13-59. FCM Read Data Timing with Extended Hold Time (for TRLX = 0, EHTR = 1, RST = 1, SCY = 1)**

### 13.4.3.4 FCM Boot Chip-Select Operation

Boot chip-select operation allows address decoding for a boot ROM before system initialization. $\overline{\text{LCS0}}$ is the boot chip-select output; its operation differs from other external chip-select outputs after a system reset. When the core begins accessing memory after system reset, $\overline{\text{LCS0}}$ is asserted initially to load a 4-Kbyte boot block into the FCM buffer RAM, but core instruction fetches occur from the buffer RAM.

#### 13.4.3.4.1 FCM Bank 0 Reset Initialization

The boot chip-select also provides a programmable port size, which is configured during reset. The boot chip-select does not provide write protection. $\overline{\text{LCS0}}$ operates this way until the first write to OR0 and it can be used as any other chip-select register after the preferred address range is loaded into BR0. After the first write to OR0, the boot chip-select can be restarted only with a hardware reset. Table 13-38 describes the initial values of the boot bank in the memory controller.

**Table 13-38. Boot Bank Field Values after Reset for FCM as Boot Controller**

| Register | Field | Setting |
|---|---|---|
| BR0 | BA<br>PS<br>DECC<br>WP<br>MSEL<br>ATOM<br>V | 0000_0000_0000_0000_0<br>From cfg_rom_loc<br><br>0<br>001<br>00<br>0 |

Table 13-38. Boot Bank Field Values after Reset for FCM as Boot Controller (continued)

| Register | Field | Setting |
|---|---|---|
| OR0 | AM | 0000_0000_0000_0000_0 |
| | BCTLD | 0 |
| | PGS | From cfg_rom_loc |
| | CSCT | 1 |
| | CST | 1 |
| | CHT | 1 |
| | RST | 1 |
| | SCY | From por_cfg_scy[1:3] |
| | TRLX | 1 |
| | EHTR | 1 |

### 13.4.3.4.2 Boot Block Loading into the FCM Buffer RAM

If FCM is selected as the boot ROM controller from power-on-reset configuration, eLBC will automatically load from bank 0 a single 4 Kbyte page of boot code into the FCM buffer RAM during $\overline{\text{HRESET}}$ (See Section 4.4.3.6, “Boot ROM Location.”). The CPU can execute boot code directly from the FCM buffer RAM, but must ensure that any further data read from the NAND Flash EEPROM is transferred under software control in order to continue the bootstrap process.

Since OR0[AM] is initially cleared during reset, all CPU fetches to eLBC will access the FCM buffer RAM, which appears in the memory map as a 4-Kbyte RAM. No NAND Flash spare regions are mapped during boot, therefore only 4 Kbytes of contiguous, main region data, loaded from the first pages of the boot block, are accessible in eLBC bank 0, as indicated in Figure 13-60.

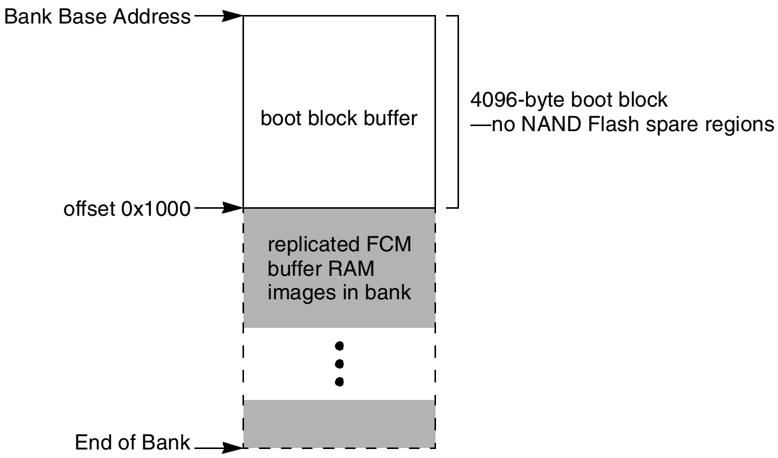


Figure 13-60. FCM Buffer RAM Memory Map During Boot Loading

The process for booting is as follows:

1. Following negation of $\overline{\text{HRESET}}$, eLBC is released from reset and commences automatic boot block loading if FCM is selected as the boot ROM location. Small-page or large-page, 8-bit NAND Flash devices can be used for boot loading when enabled with $\overline{\text{LCS0}}$. eLBC drives $\overline{\text{LFWP}}$ low during boot accesses to prevent accidental erasure of the NAND Flash boot ROM.
2. FCM starts searching for a valid boot block at block index 0.

3. FCM reads the spare regions of the first two pages of the current block, checking the bad block indication (BI) bytes to validate the block for reading. BI bytes must all hold the value 0xFF for the page to be considered readable.
   - For small-page devices, BI is a single byte read from spare region byte offset 5.
   - For large-page devices, BI is a single byte read from spare region byte offset 0.

   If either of the first two pages of the current block are marked invalid, then the boot block index is incremented by 1, and FCM repeats step 3. eLBC will continue searching for a bootable block indefinitely, therefore at least one block must be marked valid for boot loading to proceed. At the conclusion of the boot block search, the value of FBAR[BLK] points to the boot block.
4. The FCM optionally performs ECC checking at boot time depending on the configuration selected during reset. If ECC checking is enabled, the FCM recovers from the spare region the stored ECC for each 512-byte block of boot data. The boot block must be prepared with ECC protection. During ECC generation, software should use FMR[ECCM] = 0 for small-page devices, and FMR[ECCM] = 1 for large-page devices.
5. FCM performs a sequence of random-access page reads, reading entire pages from the boot block until 4 Kbytes have been saved to the FCM buffer RAM. If ECC checking is enabled, the ECC of each 512-byte region is verified and single-bit errors are corrected if possible. If FCM is unable to correct ECC errors, eLBC halts the boot process and signals an unrecoverable error by asserting the $\overline{\textit{hreset\_req}}$ signal.
6. The CPU now commences fetching instructions, in random order, from the FCM buffer RAM. This first-level boot loader typically copies a secondary boot loader into system memory, and continues booting from there. Boot software must clear FMR[BOOT] to enable normal operation of FCM.

### 13.4.4 User-Programmable Machines (UPMs)

UPMs are flexible interfaces that connect to a wide range of memory devices. At the heart of each UPM is an internal RAM array that specifies the logical value driven on the external memory control signals ($\overline{\text{LCS}}$*n*, $\overline{\text{LBS}}$[0:3] and LGPL[0:5]) for a given clock cycle. Each word in the RAM array provides bits that allow a memory access to be controlled with a resolution of up to one quarter of the external bus clock period on the byte-select and chip-select lines. A gap of 2 dead LCLK cycles is present on the UPM interface between UPM transactions.

**NOTE**

If the LGPL4/$\overline{\text{LGTA}}$/LFR$\overline{\text{B}}$/LUPWAIT/LPBSE signal is used as both an input and an output, a weak pull-up is required. Refer to the hardware specification for details regarding termination options.

Figure 13-61 shows the basic operation of each UPM.

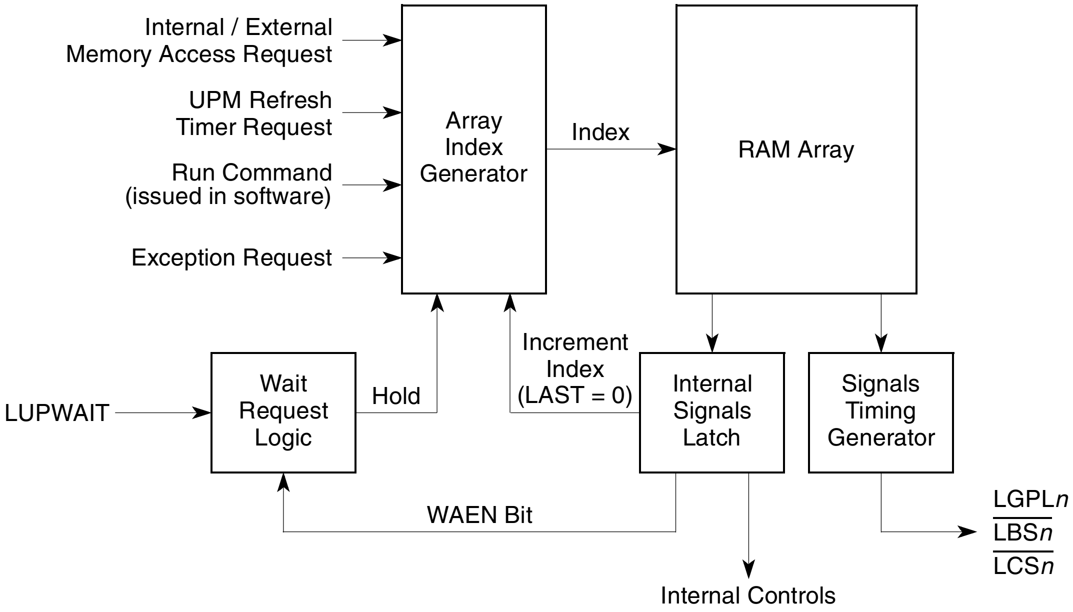


**Figure 13-61. User-Programmable Machine Functional Block Diagram**

The following events initiate a UPM cycle:

- Any internal device requests an external memory access to an address space mapped to a chip-select serviced by the UPM
- A UPM refresh timer expires and requests a transaction, such as a DRAM refresh
- A bus monitor time-out error during a normal UPM cycle redirects the UPM to execute an exception sequence

The RAM array contains 64 words of 32-bits each. The signal timing generator loads the RAM word from the RAM array to drive the general-purpose lines, byte-selects, and chip-selects. If the UPM reads a RAM word with WAEN set, the external LUPWAIT signal is sampled and synchronized by the memory controller and the current request is frozen.

### 13.4.4.1 UPM Requests

A special pattern location in the RAM array is associated with each of the possible UPM requests. An internal device's request for a memory access initiates one of the following patterns (M*x*MR[OP] = 00):

- Read single-beat pattern (RSS)
- Read burst cycle pattern (RBS)
- Write single-beat pattern (WSS)
- Write burst cycle pattern (WBS)

A UPM refresh timer request pattern initiates a refresh timer pattern (RTS).

An exception (caused by a bus monitor time-out error) occurring while another UPM pattern is running initiates an exception condition pattern (EXS).

Figure 13-62 and Table 13-39 show the start addresses of these patterns in the UPM RAM, according to cycle type. RUN commands (M*x*MR[OP] = 11), however, can initiate patterns starting at any of the 64 UPM RAM words.

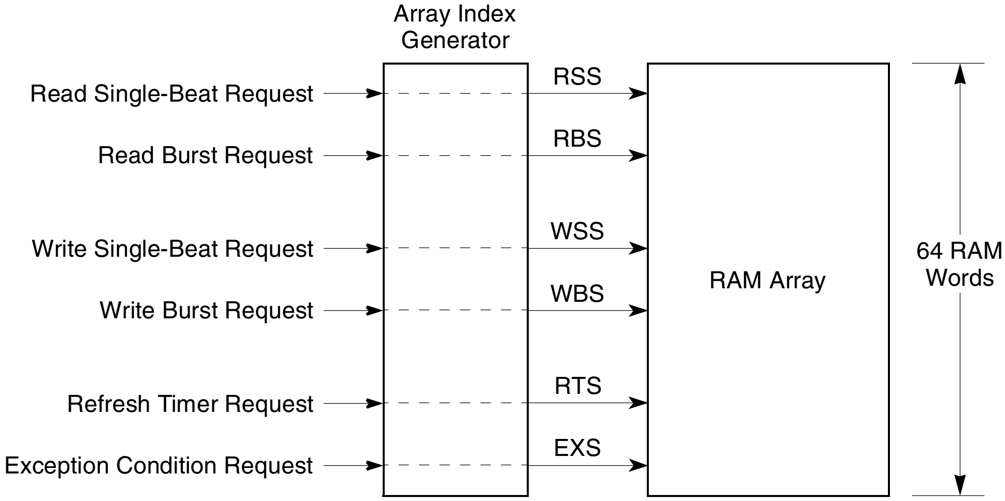


**Figure 13-62. RAM Array Indexing**

**Table 13-39. UPM Routines Start Addresses**

| UPM Routine | Routine Start Address |
|---|---|
| Read single-beat (RSS) | 0x00 |
| Read burst (RBS) | 0x08 |
| Write single-beat (WSS) | 0x18 |
| Write burst (WBS) | 0x20 |
| Refresh timer (RTS) | 0x30 |
| Exception condition (EXS) | 0x3C |

### 13.4.4.1.1 Memory Access Requests

The user must ensure that the UPM is appropriately initialized before a request occurs.

The UPM supports two types of memory reads and writes:

- A single-beat transfer transfers one operand consisting of up to a single word (dependent on port size). A single-beat cycle starts with one transfer start and ends with one transfer acknowledge.
- A burst transfer transfers exactly 4 double words regardless of port size. For 32-bit accesses, the burst cycle starts with one transfer start but ends after eight transfer acknowledges, whereas an 8-bit device requires 32 transfer acknowledges.

The user must ensure that patterns for single-beat transfers contain one, and only one, transfer acknowledge (UTA bit in RAM word set high) and for a burst transfer, contain the exact number of transfer acknowledges required.

Any transfers that do not naturally fit single or burst transfers are synthesized as a series of single transfers. These accesses are treated by the UPM as back-to-back, single-beat transfers. Burst transfers can also be inhibited by setting OR*n*[BI]. Burst performance can be achieved by ensuring that UPM transactions are 32-byte aligned with a transaction size being some multiple of 32-bytes, which is a natural fit for cache-line transfers, for example.

#### 13.4.4.1.2 UPM Refresh Timer Requests

Each UPM contains a refresh timer that can be programmed to generate refresh service requests of a particular pattern in the RAM array. Figure 13-63 shows the clock division hardware associated with memory refresh timer request generation. The UPM refresh timer register (LURT) defines the period for the timers associated with all three UPMs.

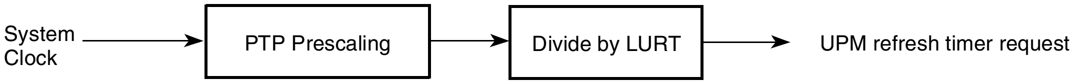


**Figure 13-63. Memory Refresh Timer Request Block Diagram**

By default, all local bus refreshes are performed using the refresh pattern of UPMA. This means that if refresh is required, MAMR[RFEN] must be set. It also means that only one refresh routine should be programmed and be placed in UPMA, which serves as the refresh executor. Any banks assigned to a UPM are provided with the common UPMA refresh pattern if the RFEN bit of the corresponding UPM is set, concurrently. UPMA assigned banks, therefore, always receive refresh services when MAMR[RFEN] is set, while UPMB and UPMC assigned banks also receive (the same) refresh services if the corresponding MxMR[RFEN] bits are set. In this scenario, more than one chip select may assert at the same time, as refresh pattern runs for all banks assigned to UPM with RFEN bit set.

#### 13.4.4.1.3 Software Requests—RUN Command

Software can start a request to the UPM by issuing a RUN command to the UPM. Some memory devices have their own signal handshaking protocol to put them into special modes, such as self-refresh mode.

For these special cycles, the user creates a special RAM pattern that can be stored in any unused areas in the UPM RAM. Then a RUN command is used to run the cycle. The UPM runs the pattern beginning at the specified RAM location until it encounters a RAM word with its LAST bit set. The RUN command is issued by setting M*x*MR[OP] = 11 and accessing UPM*n* memory region with any write transaction that hits the corresponding UPM machine. M*x*MR[MAD] determines the starting address in the RAM array for the pattern.

Note that transfer acknowledges (UTA bit in the RAM word) are ignored for software (RUN command) requests, and hence the LAD signals remain high-impedance unless the normal initial LALE occurs or the RUN pattern causes assertion of LALE to occur on changes to the RAM word AMX field.

#### 13.4.4.1.4 Exception Requests

When the eLBC under UPM control initiates an access to a memory device and an exception occurs (bus monitor time-out), the UPM provides a mechanism by which memory control signals can meet the device's timing requirements without losing data. The mechanism is the exception pattern that defines how the UPM negates its signals in a controlled manner.

#### 13.4.4.2 Programming the UPMs

The UPM is a micro sequencer that requires microinstructions or RAM words to generate signal timings for different memory cycles. Follow these steps to program the UPMs:

1. Set up BR*n* and OR*n* registers.
2. Write patterns into the RAM array.
3. Program MRTPR, LURT and MAMR[RFEN] if refresh is required.
4. Program M*x*MR.

Patterns are written to the RAM array by setting M*x*MR[OP] = 01 and accessing the UPM with any write transaction that hits the relevant chip select. The entire array is thus programmed by an alternating series of writes: to MDR (RAM word to be written) each time followed by a read from MDR and then followed by a (dummy) write transaction to the relevant UPM assigned bank. A read from MDR is required to ensure that the MDR update has occurred prior to the (dummy) write transaction.

RAM array contents may also be read for debug purposes, for example, by alternating dummy read transactions, each time followed by reads of MDR (when M*x*MR[OP] = 10).

**NOTE**

M*x*MR / MDR registers should not be updated while dummy read/write access is still in progress. If the MxMR[MAD] is incremented then the previous dummy transaction is already completed.

In order to enforce proper ordering between updates to the M*x*MR/MDR register and the dummy accesses to the UPM memory region, two rules must be followed:

1. Since the result of any update to the M*x*MR/MDR register must be in effect before the dummy read or write to the UPM region, a write to M*x*MR/MDR should be followed immediately by a read of M*x*MR/MDR.
2. The UPM memory region should have the same MMU settings as the memory region containing the M*x*MR configuration register; both should be mapped by the MMU as cache-inhibited and guarded. This prevents the CPU from re-ordering a read of the UPM memory around the read of M*x*MR. Once the programming of the UPM array is complete the MMU setting for the associated address range can be set to the proper mode for normal operation, such as cacheable and copyback.

For proper signalling, the following guidelines must be followed while programming UPM RAM words:

- For UPM reads, program UTA and LAST in the same or consecutive RAM words.
- For UPM burst reads, program last UTA and LAST in the same or consecutive RAM words.
- For UPM writes, program UTA and LAST in the same RAM word.
- For UPM burst writes, program last UTA and LAST in the same RAM word.

#### 13.4.4.2.1 UPM Programming Example (Two Sequential Writes to the RAM Array)

The following example further illustrates the steps required to perform two writes to the RAM array at non-sequential addresses assuming that the relevant BR*n* and OR*n* registers have been previously set up:

1. Program M*x*MR for the first write (with the desired RAM array address).
2. Write pattern/data to MDR to ensure that the MxMR has already been updated with the desired configuration.
3. Read MDR to ensure that the MDR has already been updated with the desired pattern. (Or, read MxMR register if step 2 is not performed.)
4. Perform a dummy write transaction.
5. Read/check M*x*MR[MAD]. If incremented, the previous dummy write transaction is completed; proceed to step 6. Repeat step 5 until incremented.
6. Program M*x*MR for the second write with the desired RAM array address.
7. Write pattern/data to MDR to ensure that the M*x*MR has already been updated with the desired configuration.
8. Read MDR to ensure that the MDR has already been updated with the desired pattern.
9. Perform a dummy write transaction.
10. Read/check M*x*MR[MAD]. If incremented, the previous dummy write transaction is completed.

Note that if step 1 (or 6) and 2 (or 7) are reversed, step 3 (or 8) is replaced by the following:

- Read M*x*MR to ensure that the M*x*MR has already been updated with the desired configuration.

#### 13.4.4.2.2 UPM Programming Example (Two Sequential Reads from the RAM Array)

RAM array contents may also be read for debug purposes, for example, by alternating dummy read transactions, each time followed by reads of MDR (M*x*MR[OP] = 0b10). The following example further illustrates the steps required to perform two reads from the RAM array at non-sequential addresses assuming that the relevant BR*n* and OR*n* registers have been previously set up:

1. Program M*x*MR for the first read with the desired RAM array address.
2. Read M*x*MR to ensure that the M*x*MR has already been updated with the desired configuration, such as RAM array address.
3. Perform a dummy read transaction.
4. Read/check M*x*MR[MAD]. If incremented, the previous dummy read transaction is completed; proceed to step 5. Repeat step 4 until incremented.
5. Read MDR.
6. Program M*x*MR for the second read with the desired RAM array address.
7. Read M*x*MR to ensure that the M*x*MR has already been updated with the desired configuration, such as RAM array address.
8. Perform a dummy read transaction.
9. Read/check M*x*MR[MAD]. If incremented, the previous dummy read transaction is completed; proceed to step 10. Repeat step 9 until incremented.
10. Read MDR.

### 13.4.4.3 UPM Signal Timing

RAM word fields specify the value of the various external signals at a granularity of up to four values for each bus clock cycle. The signal timing generator causes external signals to behave according to timing specified in the current RAM word. ach bit in the RAM word relating to LCS*n* and LBS timing specifies the value of the corresponding external signal at each quarter phase of the bus clock.

The division of UPM bus cycles into phases is shown in Figure 13-64.

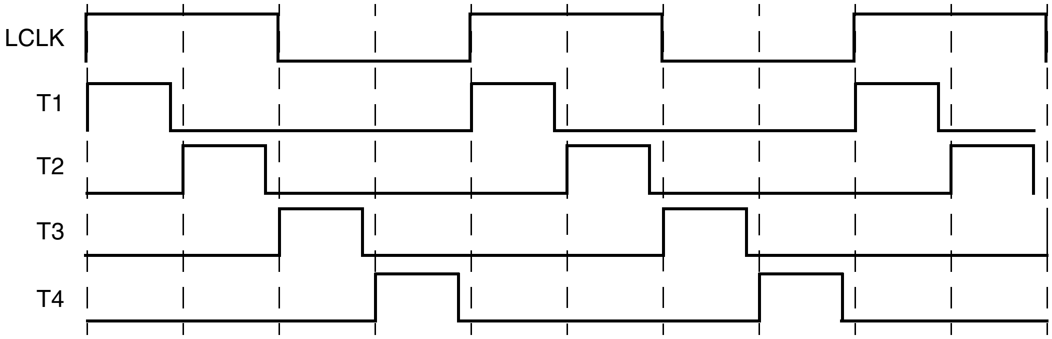


**Figure 13-64. UPM Clock Scheme**

### 13.4.4.4 RAM Array

The RAM array for each UPM is 64 locations deep and 32 bits wide, as shown in Figure 13-65. The signals at the bottom of the figure are UPM outputs. The selected LCS*n* is for the bank that matches the current address. The selected LBS is for the byte lanes read or written by the access.

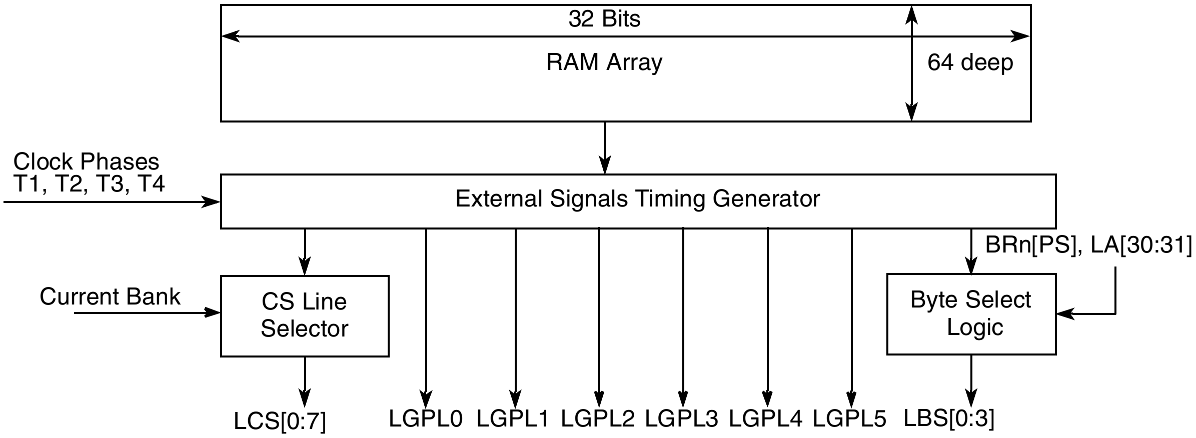


**Figure 13-65. RAM Array and Signal Generation**

#### 13.4.4.4.1 RAM Words

The RAM word is a 32-bit microinstruction stored in one of 64 locations in the RAM array. It specifies timing for external signals controlled by the UPM. Figure 13-39 shows the RAM word fields. he CST*n* and BST*n* bits determine the state of UPM signals LCS*n* and LBS[0:3] at each quarter phase of the bus clock.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | CST1 | CST2 | CST3 | CST4 | BST1 | BST2 | BST3 | BST4 | G0L | | G0H | | G1T1 | G1T3 | G2T1 | G2T3 |
| Reset | All zeros | | | | | | | | | | | | | | | |

| | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | G3T1 | G3T3 | G4T1/<br>DLT3 | G4T3/<br>WAEN | G5T1 | G5T3 | REDO | | LOOP | EXEN | AMX | | NA | UTA | TODT | LAST |
| Reset | All zeros | | | | | | | | | | | | | | | |

**Figure 13-66. RAM Word Fields**

Table 13-40 contains descriptions of the RAM word fields.

**Table 13-40. RAM Word Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | CST1 | Chip select timing 1. Defines the state (0 or 1) of $\overline{\mathrm{LCS}n}$ during bus clock quarter phase 1. |
| 1 | CST2 | Chip select timing 2. Defines the state (0 or 1) of $\overline{\mathrm{LCS}n}$ during bus clock quarter phase 2. |
| 2 | CST3 | Chip select timing 3. Defines the state (0 or 1) of $\overline{\mathrm{LCS}n}$ during bus clock quarter phase 3. |
| 3 | CST4 | Chip select timing 4. Defines the state (0 or 1) of $\overline{\mathrm{LCS}n}$ during bus clock quarter phase 4. |
| 4 | BST1 | Byte select timing1. Defines the state (0 or 1) of $\overline{\mathrm{LBS}}$ during bus clock quarter phase 1. |
| 5 | BST2 | Byte select timing 2. Defines the state (0 or 1) of $\overline{\mathrm{LBS}}$ during bus clock quarter phase 2. |
| 6 | BST3 | Byte select timing 3. Defines the state (0 or 1) of $\overline{\mathrm{LBS}}$ during bus clock quarter phase 3. |
| 7 | BST4 | Byte select timing 4. Defines the state (0 or 1) of $\overline{\mathrm{LBS}}$ during bus clock quarter phase 4. |
| 8–9 | G0L | General purpose line 0 lower. Defines the state of LGPL0 during the bus clock quarter phases 1 and 2 (first half phase).<br>00 Value defined by M*x*MR[G0CL]<br>01 Reserved<br>10 0<br>11 1 |
| 10–11 | G0H | General purpose line 0 higher. Defines the state of LGPL0 during the bus clock quarter phases 3 and 4 (second half phase).<br>00 Value defined by M*x*MR[G0CL]<br>01 Reserved<br>10 0<br>11 1 |
| 12 | G1T1 | General purpose line 1 timing 1. Defines the state (0 or 1) of LGPL1 during bus clock quarter phases 1 and 2 (first half phase). |
| 13 | G1T3 | General purpose line 1 timing 3. Defines the state (0 or 1) of LGPL1 during bus clock quarter phases 3 and 4 (second half phase) |
| 14 | G2T1 | General purpose line 2 timing 1. Defines state (0 or 1) of LGPL2 during bus clock quarter phases 1 and 2 (first half phase). |
| 15 | G2T3 | General purpose line 2 timing 3. Defines the state (0 or 1) of LGPL2 during bus clock quarter phases 3 and 4 (second half phase). |

**Table 13-40. RAM Word Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 16 | G3T1 | General purpose line 3 timing 1. Defines the state (0 or 1) of LGPL3 during bus clock quarter phases 1 and 2 (first half phase). |
| 17 | G3T3 | General purpose line 3 timing 3. Defines the state (0 or 1) of LGPL3 during bus clock quarter phases 3 and 4 (second half phase). |
| 18 | G4T1/DLT3 | General purpose line 4 timing 1/delay time 3. The function of this bit is determined by M*x*MR[GPL4].<br>If M*x*MR[GPL4] = 0 and LGPL4/LUPWAIT pin functions as an output (LGPL4), G4T1/DLT3 defines the state (0 or 1) of LGPL4 during bus clock quarter phases 1 and 2 (first half phase).<br>If M*x*MR[GPL4] = 1 and LGPL4/LUPWAIT functions as an input (LUPWAIT), if a read burst or single read is executed, G4T1/DLT3 defines the sampling of the data bus as follows:<br>0 In the current word, the data bus should be sampled at the start of bus clock quarter phase 1 of the next bus clock cycle.<br>1 In the current word, the data bus should be sampled at the start of bus clock quarter phase 3 of the current bus clock cycle. |
| 19 | G4T3/WAEN | General purpose line 4 timing 3/wait enable. Bit function is determined by M*x*MR[GPL4].<br>If M*x*MR[GPL4] = 0 and LGPL4/LUPWAIT pin functions as an output (LGPL4), G4T3/WAEN defines the state (0 or 1) of LGPL4 during bus clock quarter phases 3 and 4 (second half phase).<br>If M*x*MR[GPL4] = 1 and LGPL4/LUPWAIT functions as an input (LUPWAIT), G4T3/WAEN is used to enable the wait mechanism:<br>0 LUPWAIT detection is disabled.<br>1 LUPWAIT is enabled. If LUPWAIT is detected as being asserted, a freeze in the external signals logical values occurs until LUPWAIT is detected as being negated. |
| 20 | G5T1 | General purpose line 5 timing 1. Defines the state (0 or 1) of LGPL5 during bus clock quarter phases 1 and 2 (first half phase). |
| 21 | G5T3 | General purpose line 5 timing 3. Defines the state (0 or 1) of LGPL5 during bus clock quarter phases 3 and 4 (second half phase). |
| 22–23 | REDO | Redo current RAM word. Defines the number of times to execute the current RAM word.<br>00 Once (normal operation)<br>01 Twice<br>10 Three times<br>11 Four times |
| 24 | LOOP | Loop start/end. The first RAM word in the RAM array where LOOP is 1 is recognized as the loop start word. The next RAM word where LOOP is 1 is the loop end word. RAM words between, and including the start and end words, are defined as part of the loop. The number of times the UPM executes this loop is defined in the corresponding loop fields of the M*x*MR.<br>0 The current RAM word is not the loop start word or loop end word.<br>1 The current RAM word is the start or end of a loop.<br>**Note:** AMX must not change values in any RAM word which begins a loop |

Table 13-40. RAM Word Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 25 | EXEN | Exception enable. Allows branching to an exception pattern at the exception start address (EXS). When an internal bus monitor time-out exception is recognized and EXEN in the RAM word is set, the UPM branches to the special exception start address (EXS) and begins operating as the pattern defined there specifies.<br>The user should provide an exception pattern to negate signals controlled by the UPM in a controlled fashion. For DRAM control, a handler should negate RAS and CAS to prevent data corruption. If EXEN = 0, exceptions are ignored by UPM (but not by local bus) and execution continues. After the UPM branches to the exception start address, it continues reading until the LAST bit is set in the RAM word.<br>0 The UPM continues executing the remaining RAM words, ignoring any internal bus monitor time-out.<br>1 The current RAM word allows a branch to the exception pattern after the current cycle if an exception condition is detected. |
| 26–27 | AMX | Address multiplexing. Determines the source of LAD during an LALE phase. Any change in the AMX field initiates a new LALE (address) phase.<br>00 LAD (and/or in conjunction with LA) is the non-multiplexed address. For example, column address.<br>01 Reserved<br>10 LAD (and/or in conjunction with LA) is driven with the multiplexed address according to M*x*MR[AM]. For example, row address. See Section 13.4.4.4.7, “Address Multiplexing (AMX)” for more information.<br>11 LAD (and/or in conjunction with LA) is driven with the contents of MAR. Used, for example, to initialize a mode.<br>**Note:** AMX must not change values in any RAM word which begins a loop.<br>**Note:** Source ID debug mode is only supported for the AMX = 00 setting. |
| 28 | NA | Next burst address. Determines when the address is incremented during a burst access.<br>0 The address increment function is disabled.<br>1 The address is incremented in the next cycle. In conjunction with the BR*n*[PS], the increment value of LA*n* is 1, 2, or 4 for port sizes of 8 bits, 16 bits, and 32 bits, respectively. |
| 29 | UTA | UPM transfer acknowledge. Indicates assertion of transfer acknowledge in the current cycle.<br>0 Transfer acknowledge is not asserted in the current cycle.<br>1 Transfer acknowledge is asserted in the current cycle.<br>In case of UPM writes, program UTA and LAST in same RAM word.<br>In case of UPM reads, program UTA and LAST in consecutive or same RAM words. |
| 30 | TODT | Turn-on disable timer. The disable timer associated with each UPM allows a minimum time to be guaranteed between two successive accesses to the same memory bank. This feature is critical when DRAM requires a RAS precharge time. TODT turns the timer on to prevent another UPM access to the same bank until the timer expires.The disable timer period is determined in M*x*MR[DS*n*]. The disable timer does not affect memory accesses to different banks. Note that TODT must be set together with LAST, otherwise it is ignored.<br>0 The disable timer is turned off.<br>1 The disable timer for the current bank is activated preventing a new access to the same bank (when controlled by the UPMs) until the disable timer expires. For example, precharge time. |

Table 13-40. RAM Word Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 31 | LAST | Last word. When LAST is read in a RAM word, the current UPM pattern terminates and control signal timing set in the RAM word is applied to the current (and last) cycle. However, if the disable timer is activated and the next access is to the same bank, execution of the next UPM pattern is held off and the control signal values specified in the last word are extended in duration for the number of clock cycles specified in M*x*MR[DS*n*].<br>0 The UPM continues executing RAM words.<br>1 Indicates the last RAM word in the program. The service to the UPM request is done after this cycle concludes.<br>In case of UPM writes, program UTA and LAST in same RAM word.<br>In case of UPM reads, program UTA and LAST in consecutive or same RAM words. |

### 13.4.4.4.2 Chip-Select Signal Timing (CST*n*)

If BR*n*[MSEL] of the accessed bank selects a UPM on the currently requested cycle, the UPM manipulates the $\overline{\text{LCS}}n$ for that bank with timing as specified in the UPM RAM word CST*n* fields. The selected UPM affects only the assertion and negation of the appropriate $\overline{\text{LCS}}n$ signal. The state of the selected $\overline{\text{LCS}}n$ signal of the corresponding bank depends on the value of each CST*n* bit. Figure 13-67 shows how UPMs control $\overline{\text{LCS}}n$ signals.

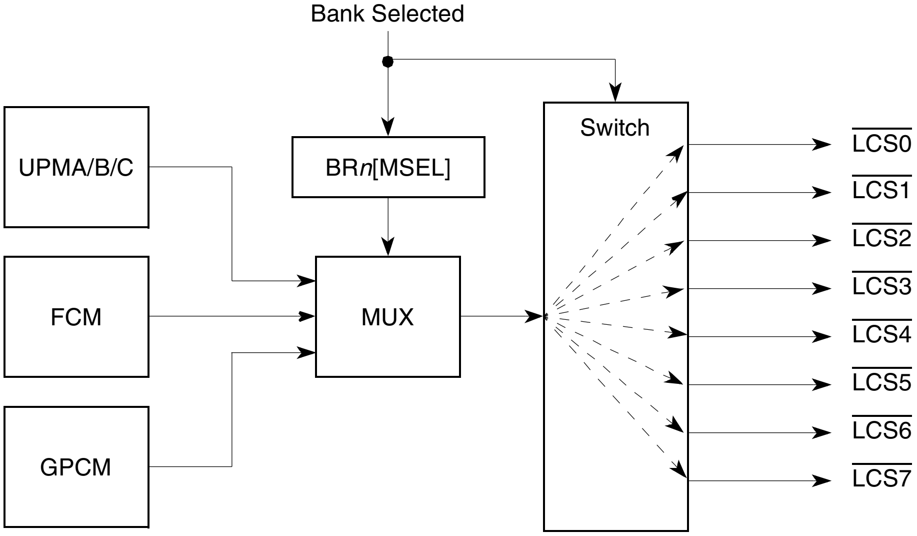


Figure 13-67. $\overline{\text{LCS}}n$ Signal Selection

### 13.4.4.4.3 Byte Select Signal Timing (BST*n*)

If BR*n*[MSEL] of the accessed memory bank selects a UPM on the currently requested cycle, the selected UPM affects the assertion and negation of the appropriate $\overline{\text{LBS}}$[0:3] signal. The timing of all four byte-select signals is specified in the RAM word. However, $\overline{\text{LBS}}$[0:3] are also controlled by the port size of the accessed bank, the number of bytes to transfer, and the address accessed. Figure 13-68 shows how UPMs control $\overline{\text{LBS}}$[0:3].

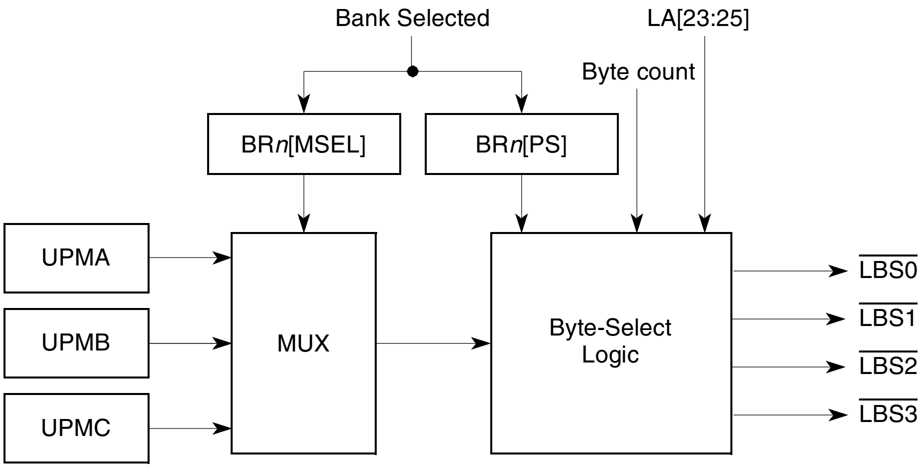


**Figure 13-68. $\overline{\text{LBS}}$ Signal Selection**

The uppermost byte select ($\overline{\text{LBS0}}$), when asserted, indicates that LAD[0:7] contains valid data during a cycle. Likewise, $\overline{\text{LBS1}}$ indicates that LAD[8:15] contain valid data, $\overline{\text{LBS2}}$ indicates that LAD[16:23] contains valid data, and $\overline{\text{LBS3}}$ indicates that LAD[24:31] contain valid data. For a UPM refresh timer request, all $\overline{\text{LBS}}$[0:3] signals are asserted/negated by the UPM according to the refresh pattern only. Following any internal bus monitor exception, the $\overline{\text{LBS}}$[0:3] signals are negated regardless of the exception handling provided by any UPM exception pattern to prevent spurious writes to external RAM.

### 13.4.4.4.4 General-Purpose Signals (G*n*T*n*, GO*n*)

The general-purpose signals (LGPL[0:5]) each have two bits in the RAM word that define the logical value of the signal to be changed at the rising edge of the bus clock and/or at the falling edge of the bus clock. LGPL0 offers enhancements beyond the other LGPL*n* lines.

LGPL0 can be controlled by an address line specified in M*x*MR[G0CL]. To use this feature, G0H and G0L should be set in the RAM word. For example, for a SIMM with multiple banks, this address line can be used to switch between internal memory device banks.

### 13.4.4.4.5 Loop Control (LOOP)

The LOOP bit in the RAM word specifies the beginning and end of a set of UPM RAM words that are to be repeated. The first time LOOP = 1, the memory controller recognizes it as a loop start word and loads the memory loop counter with the corresponding contents of the loop field shown in Table 13-41. The next RAM word for which LOOP = 1 is recognized as a loop end word. When it is reached, the loop counter is decremented by one.

Continued loop execution depends on the loop counter. If the counter is not zero, the next RAM word executed is the loop start word. Otherwise, the next RAM word executed is the one after the loop end word. Loops can be executed sequentially but cannot be nested. Also, special care must be taken:

- LAST and LOOP must not be set together.
- Loop start word should not have an AMX change with regard to the previous word.

Table 13-41. M*x*MR Loop Field Use

| Request Serviced | Loop Field |
|---|---|
| Read single-beat cycle | RLF |
| Read burst cycle | RLF |
| Write single-beat cycle | WLF |
| Write burst cycle | WLF |
| Refresh timer expired | TLF |
| RUN command | RLF |

#### 13.4.4.4.6 Repeat Execution of Current RAM Word (REDO)

The REDO function is useful for wait-state insertion in a long UPM routine that would otherwise need too many RAM words. Setting the REDO bits of the RAM word to a nonzero value causes the UPM to re-execute the current RAM word up to three more times, as defined in the REDO field of the current RAM word.

Special care must be taken in the following cases:

- When UTA and REDO are set together, TA is asserted the number of times specified by the REDO function.
- When NA and REDO are set together, the address is incremented the number of times specified by the REDO function.
- When LOOP and REDO are set together, the loop mechanism works as usual and the line is repeated according to the REDO function.
- LAST and REDO must not be set together.
- REDO should not be used within the exception routine.

#### 13.4.4.4.7 Address Multiplexing (AMX)

Address lines can be controlled by the user-provided pattern in the UPM. The address multiplex (AMX) bits in the RAM word can choose between driving the transaction address (AMX = 00), driving it according to the multiplexing specified by the M*x*MR[AM] field (AMX = 10), or driving the contents of MAR (AMX = 11) on the address signals. The next address (NA) bit of the RAM word does not affect LA signals, unless AMX = 00 and chooses the column address for NA = 1.

In all cases, LA[27:31] of the eLBC are driven by the five lsbs of the address selected by AMX, regardless of whether the next address (NA) bit of the RAM word is used to increment the current address. The effect of NA = 1 is visible only when AMX = 00 chooses the column address.

Table 13-42 shows how the RAM word AMX bits and M*x*MR[AM] settings can be used to affect row × column address multiplexing on the LAD[16:31] signals. When AMX = 10, LAD[0:15] are driven low during an address phase.

**Table 13-42. UPM Address Multiplexing**

| | msb | Internal Transaction Address | | | | | | | | | | | | | | | | | | | | | | | | | | | | | lsb |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** | **25** | **26** | **27** | **28** | **29** | **30** | **31** |
| AMX = 10<br>MxMR[AM] = 000<br>(Row) | | | | | | | | | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | | | | | | | | |
| AMX = 00<br>(Col) | | | | | | | | | | | | | | | | | | | | | | | | | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| AMX = 10<br>MxMR[AM] = 001<br>(Row) | | | | | | | | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | | | | | | | | | |
| AMX = 00<br>(Col) | | | | | | | | | | | | | | | | | | | | | | | | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| AMX = 10<br>MxMR[AM] = 010<br>(Row) | | | | | | | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | | | | | | | | | | |
| AMX = 00<br>(Col) | | | | | | | | | | | | | | | | | | | | | | | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| AMX = 10<br>MxMR[AM] = 011<br>(Row) | | | | | | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | | | | | | | | | | | |
| AMX = 00<br>(Col) | | | | | | | | | | | | | | | | | | | | | | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| AMX = 10<br>MxMR[AM] = 100<br>(Row) | | | | | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | | | | | | | | | | | | |
| AMX = 00<br>(Col) | | | | | | | | | | | | | | | | | | | | | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| AMX = 10<br>MxMR[AM] = 101<br>(Row) | | | | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | | | | | | | | | | | | | |
| AMX = 00<br>(Col) | | | | | | | | | | | | | | | | | | | | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| AMX = 10<br>MxMR[AM] = 110 | Reserved | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| AMX = 10<br>MxMR[AM] = 111 | Reserved | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Note that any change to the AMX field from one RAM word to the next RAM word executed results in an address phase on the {LAD*n*, LA*n*} bus with the assertion of LALE for the number of cycles set for LALE in the OR*n* and LCRR registers. The LGPL[0:5] signals maintain the value specified in the RAM word during the LALE phase.

**NOTE**

AMX must not change values in any RAM word which begins a loop.

#### 13.4.4.4.8 Data Valid and Data Sample Control (UTA)

When a read access is handled by the UPM, and the UTA bit is 1 (data is to be sampled by the eLBC), the value of the DLT3 bit in the same RAM word, in conjunction with M*x*MR[GPL4], determines when the data input is sampled by the eLBC as follows:

- If M*x*MR[GPL4] = 1 (G4T4/DLT3 functions as DLT3) and DLT3 = 1 in the RAM word, data is latched on the falling edge of the bus clock instead of the rising edge. The eLBC samples the data on the next falling edge of the bus clock, which is during the middle of the current bus cycle. This feature should be used only in systems without external synchronous bus devices that require mid-cycle sampling.

- If M*x*MR[GPL4] = 0 (G4T4/DLT3 functions as G4T4), or if M*x*MR[GPL4] = 1 but DLT3 = 0 in the RAM word, data is latched on the rising edge of the bus clock, which occurs at the end of the current bus clock cycle (normal operation).

Figure 13-69 shows how data sampling is controlled by the UPM.

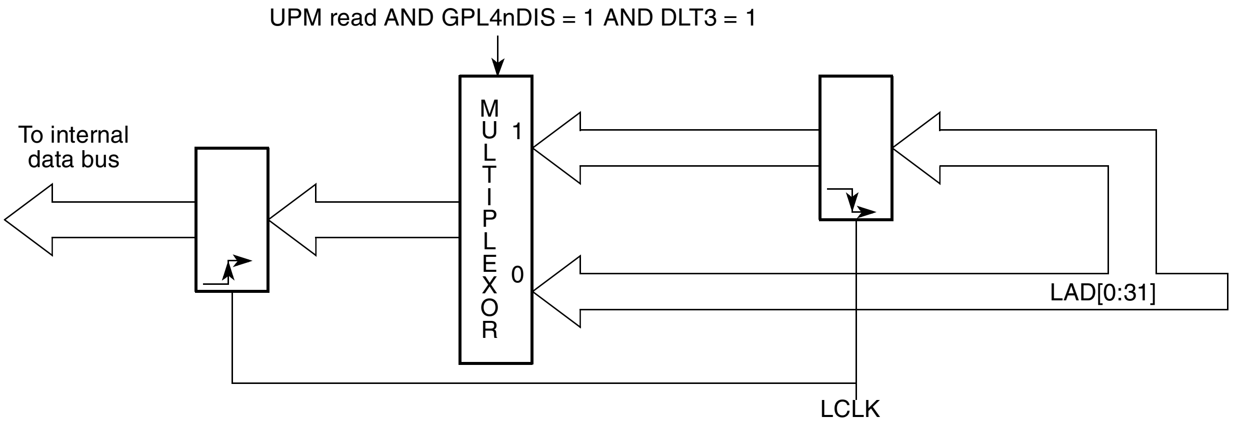


**Figure 13-69. UPM Read Access Data Sampling**

### 13.4.4.4.9 LGPL[0:5] Signal Negation (LAST)

When the LAST bit is read in a RAM word, the current UPM pattern is terminated at the end of the current cycle. On the next cycle (following LAST) all the UPM signals are negated unconditionally (driven to logic 1), unless there is a back-to-back UPM request pending. In this case, the signal values for the cycle following the one in which the LAST bit was set are taken from the first RAM word of the pending UPM routine.

In case of UPM writes, program UTA and LAST in same RAM word. In case of UPM reads, program UTA and LAST in consecutive or same RAM words.

### 13.4.4.4.10 Wait Mechanism (WAEN)

The WAEN bit in the RAM array word can be used to enable the UPM wait mechanism in selected UPM RAM words. If the UPM reads a RAM word with WAEN set, the external LUPWAIT signal is sampled and synchronized by the memory controller as if it were an asynchronous signal. The WAEN bit is ignored if LAST = 1 in the same RAM word.

Synchronization of LUPWAIT starts at the rising edge of the bus clock and takes at least 1 bus cycle to complete. If LUPWAIT is asserted and WAEN = 1 in the current UPM word, the UPM is frozen until LUPWAIT is negated. The value of external signals driven by the UPM remains as indicated in the previous RAM word. When LUPWAIT is negated, the UPM continues normal functions. Note that during WAIT cycles, the UPM does not handle data.

Figure 13-70 shows how the WAEN bit in the word read by the UPM and the LUPWAIT signal are used to hold the UPM in a particular state until LUPWAIT is negated. As the example shows, the $\overline{\text{LCS}n}$ and LGPL1 states and the WAEN value are frozen until LUPWAIT is recognized as negated. WAEN is typically set before the line that contains UTA = 1. Note that if WAEN and NA are both set in the same RAM word, NA causes the burst address to increment once as normal regardless of whether the UPM freezes.

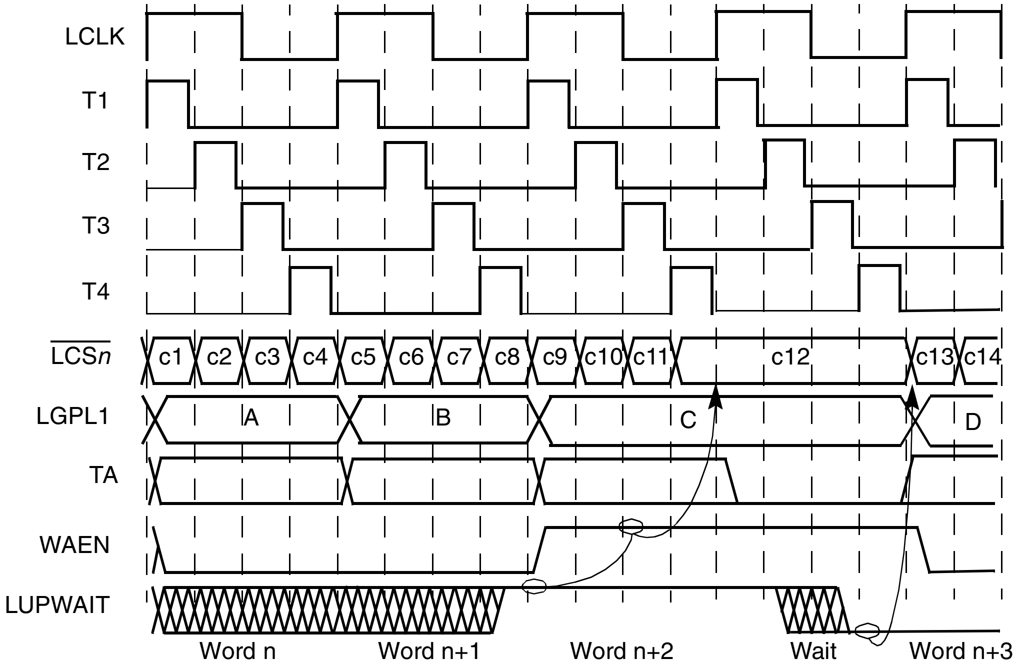


**Figure 13-70. Effect of LUPWAIT Signal**

### 13.4.4.5 Synchronous Sampling of LUPWAIT for Early Transfer Acknowledge

If LUPWAIT is to be considered an asynchronous signal, which can be asserted/negated at any time, no UPM RAM word must contain both WAEN = 1 and UTA = 1 simultaneously.

However, programming WAEN = 1 and UTA = 1 in the same RAM word, under certain conditions, allows the UPM to treat LUPWAIT as a synchronous signal, which must meet set-up and hold times in relation to the rising edge of the bus clock. The conditions are as follows:

- The PLL must be enabled, that is, LCRR[PBYP] = 0.
- DLT3 bit must be cleared in the same RAM word to avoid mid-sampling of read data.
- LBCR[LPBSE] = 0 and MXMR[GPL4] = 1
- The combination WAEN=1 and UTA=1 should be in the RAM word next to the word which gets frozen by LUPWAIT assertion. This condition limits the use of this mode to cases where the exact cycle of LUPWAIT assertion is predictable.

In this mode, as soon as UPM samples LUPWAIT negated on the rising edge of the bus clock, it immediately generates an internal transfer acknowledge, which allows a data transfer one bus clock cycle later. The generation of transfer acknowledge is early because LUPWAIT is not re-synchronized. The acknowledge occurs early or normally depending on whether the UPM was already frozen inWAIT cycles or not. This feature allows the synchronous negation of LUPWAIT to affect a data transfer, even if UTA, WAEN, and LAST are set simultaneously.

#### 13.4.4.6 Extended Hold Time on Read Accesses

Slow memory devices that take a long time to turn off their data bus drivers on read accesses should choose some non-zero combination of OR*n*[TRLX] and OR*n*[EHTR]. The next accesses after a read access to the slow memory device is delayed by the number of clock cycles specified in the OR*n* register in addition to any existing bus turnaround cycle.

## 13.5 Initialization/Application Information

### 13.5.1 Interfacing to Peripherals in Different Address Modes

This section provides guidelines for interfacing to peripherals.

#### 13.5.1.1 Multiplexed Address/Data Bus for 32-Bit Addressing

In order to reduce pins on the local bus, address and data signals are multiplexed. To build the address, an external latch is used to demultiplex and reconstruct the original address. Since the LALE signal provides the correct timing to control a standard logic latch, no external intelligence is needed. To pass data, the LAD signals can be directly connected to the data signals of the memory/peripheral.

Transactions on the local bus begin with an address phase. The eLBC drives the transaction address on the LAD signals and asserts the LALE signal to latch the address. This assertion causes address bits A[0:31] to appear on LAD[0:31]. The eLBC can then continue on into the data phase.

The eLBC supports port sizes of 8, 16, and 32 bits. When there is an access larger than the port size, the eLBC breaks up the access into smaller transactions using the non-multiplexed address signals LA*n*. For 32-bit devices, LA[30:31] are irrelevant since these address bits are implicit in the byte lanes which carry data. Similarly, for 16-bit devices, LA[30] is used and LA[31] is irrelevant; however, for 8-bit devices, LA[30:31] are necessary.

In addition, the eLBC supports burst transfers in the UPM machine. To minimize the amount of address phases needed on the local bus and to optimize the throughput, LA*n* are driven separatelyand should be used whenever a device requires the five least-significant addresses. The five least-significant address bits should not be used from LAD[27:31]. All other address bits, A[0:26], must be reconstructed through the latch, as shown in Figure 13-71.

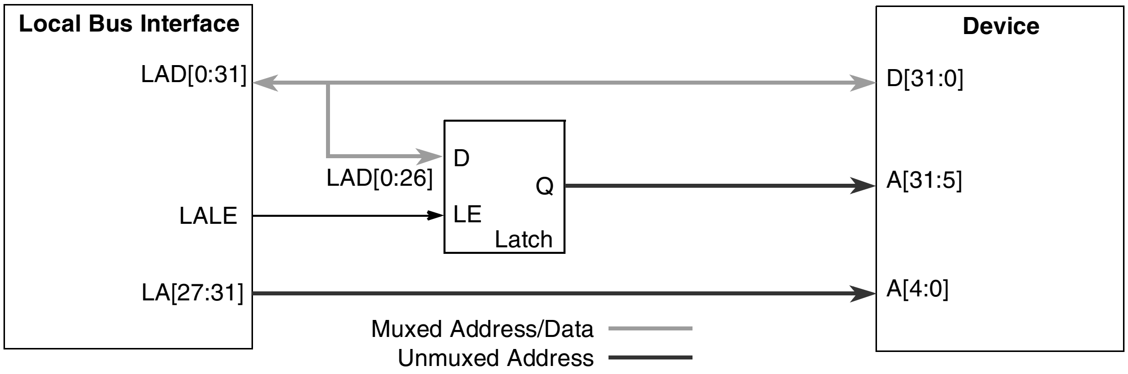


**Figure 13-71. Multiplexed Address/Data Bus for 32-Bit Addressing**

### 13.5.1.2 Peripheral Hierarchy on the Local Bus for High Bus Speeds

To achieve the highest possible bus speeds on the local bus, it is recommended to reduce the number of devices connected directly to the bus. For best results, only one bank of synchronous SRAMs should have a direct connection, and a bus demultiplexor should be used to replace separate latch and separate bus transceiver combinations. Figure 13-72 shows an example of such a hierarchy. This section is only a guideline, and the board designer must simulate the electric characteristics of the scenario to determine the maximum operating frequency.

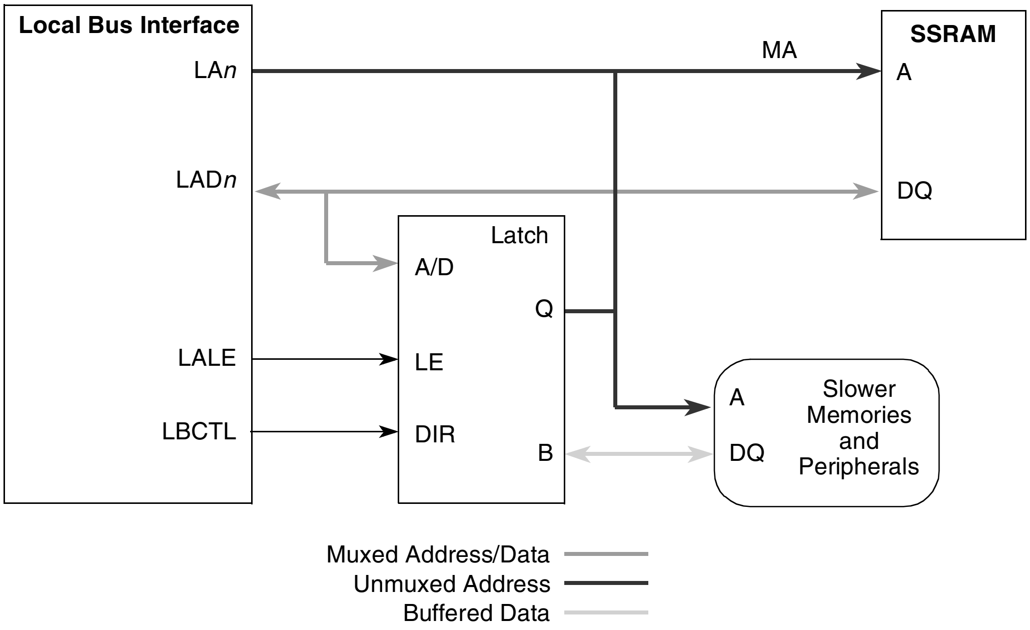


**Figure 13-72. Local Bus Peripheral Hierarchy for High Bus Speeds**

### 13.5.1.3 GPCM Timings

In case a system contains a memory hierarchy with high speed synchronous memories (synchronous SRAM) and lower speed asynchronous memories (for example, FLASH EPROM and peripherals) the GPCM-controlled memories should be decoupled by buffers to reduce capacitive loading on the bus. Those buffers have to be taken into account for the timing calculations.

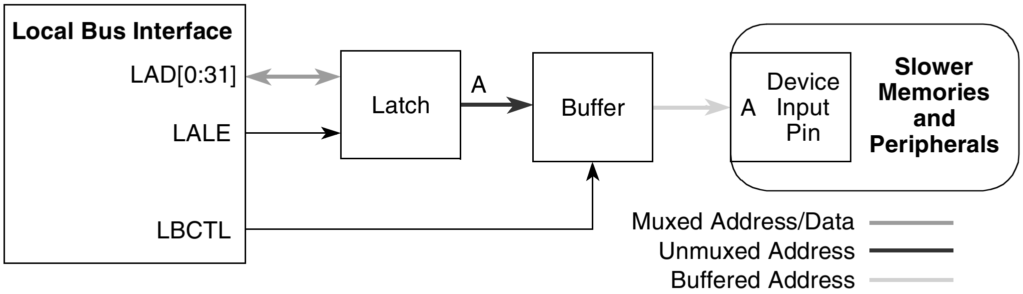


**Figure 13-73. GPCM Address Timings**

To calculate address setup timing for a slower peripheral/memory device, several parameters have to be added: propagation delay for the address latch, propagation delay for the buffer and the address setup for the actual peripheral. Typical values for the two propagation delays are in the order of 3–6 ns, so for a 133-MHz bus frequency, $\overline{\text{LCS}}$ should arrive on the order of 3 bus clocks later.

For data timings, only the propagation delay of one buffer plus the actual data setup time has to be considered.

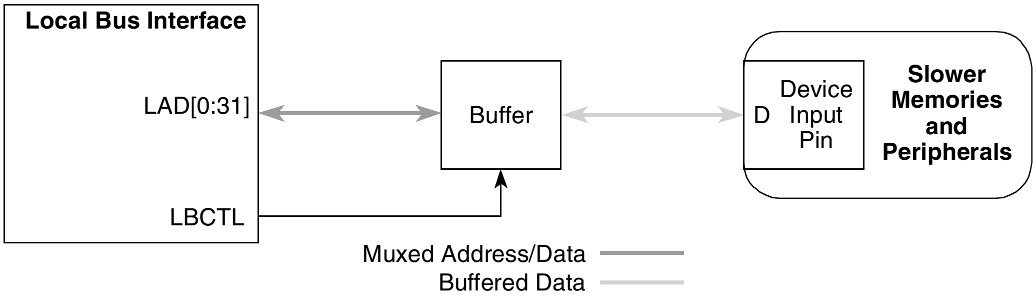


**Figure 13-74. GPCM Data Timings**

## 13.5.2 Bus Turnaround

Because the local bus uses multiplexed address and data, special consideration must be given to avoid bus contention at bus turnaround. The following cases must be examined:

- Address phase after previous read
- Read data phase after address phase
- Read-modify-write cycle for parity protected memory banks
- UPM cycles with additional address phases

The bus does not change direction for the following cases so they need no special attention:

- Continued burst after the first beat

- Write data phase after address phase
- Address phase after previous write

### 13.5.2.1 Address Phase after Previous Read

During a read cycle, the memory/peripheral drives the bus and the bus transceiver drives LAD. After the data has been sampled, the output drivers of the external device must be disabled. This can take some time; for slow devices the EHTR feature of the GPCM or the programmability of the UPM should be used to guarantee that those devices have stopped driving the bus when the eLBC memory controller ends the bus cycle.

In this case, after the previous cycle ends, LBCTL goes high and changes the direction of the bus transceiver. The eLBC then inserts a bus turnaround cycle to avoid contention. The external device has now already placed its data signals in high impedance and no bus contention will occur.

### 13.5.2.2 Read Data Phase after Address Phase

During the address phase, LAD actively drives the address and LBCTL is high, driving the bus transceivers in the same direction as during a write. After the end of the address phase, LBCTL goes low and changes the direction of the bus transceiver. The eLBC places the LAD signals in high impedance after its $t_{dis}$(LB). The LBCTL will have its new state after $t_{en}$(LB) and, because this is an asynchronous input, the transceiver starts to drive those signals after its $t_{en}$(transceiver) time. The system designer has to ensure, that [$t_{en}$(LB) + $t_{en}$(transceiver)] is larger than $t_{dis}$ (LB) to avoid bus contention.

### 13.5.2.3 Read-Modify-Write Cycle for Parity Protected Memory Banks

Principally, a read-modify-write cycle is a read cycle immediately followed by a write cycle. Because the write cycle will have a new address phase in any case, this basically is the same case as an address phase after a previous read.

### 13.5.2.4 UPM Cycles with Additional Address Phases

The flexibility of the UPM allows the user to insert additional address phases during read cycles by changing the AMX field, therefore turning around the bus during one pattern. The eLBC automatically inserts a single bus turnaround cycle if the bus (LAD) was previously high impedance for any reason, such as a read, before LALE is driven and LAD is driven with the new address. The turnaround cycle is not inserted on a write, because the bus was already driven to begin with.

However, bus contention could potentially still occur on the far side of a bus transceiver. It is the responsibility of the designer of the UPM pattern to guarantee that enough idle cycles are inserted in the UPM pattern to avoid this.

## 13.5.3 Interface to Different Port-Size Devices

The eLBC supports 8-, 16-, and 32-bit data port sizes. However, the bus requires that the portion of the data bus used for a transfer to or from a particular port size be fixed. A 32-bit port must reside on LAD[0:31], a 16-bit port must reside on LAD[0:15], and an 8-bit port must reside on LAD[0:7]. The local bus always tries to transfer the maximum amount of data on all bus cycles. Figure 13-75 shows the device connections on the data bus.

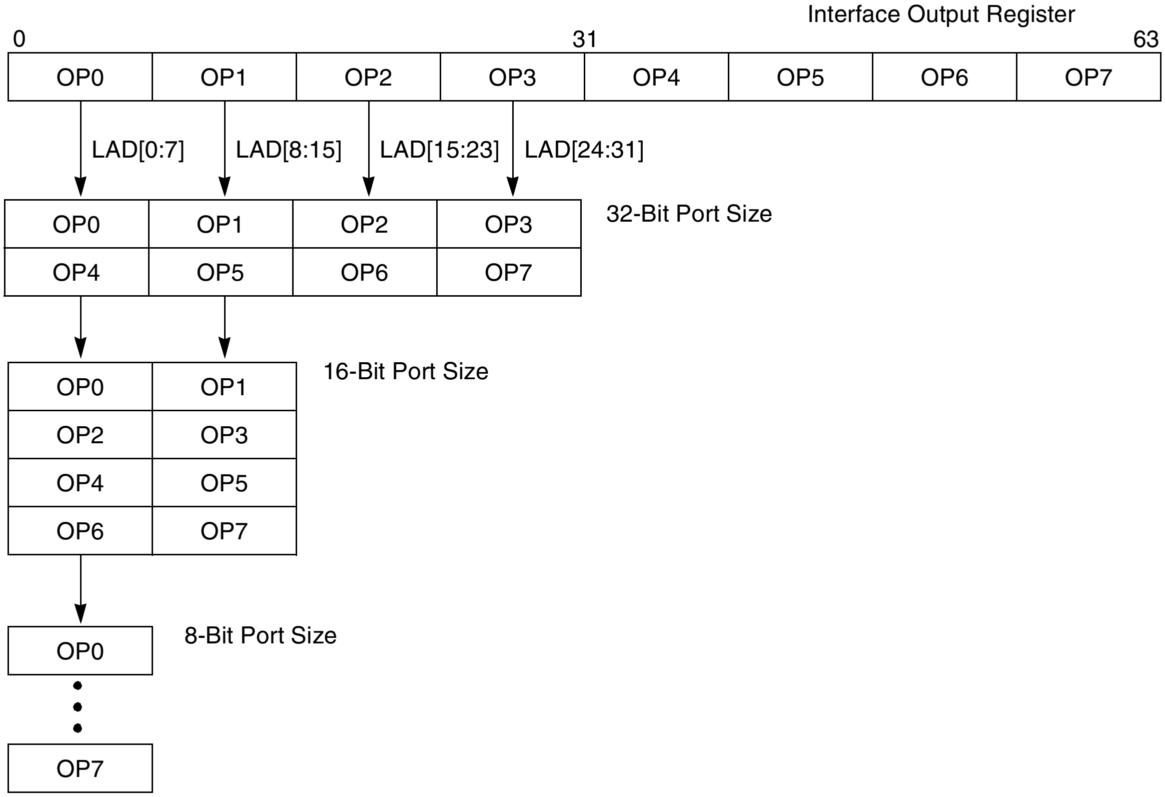


**Figure 13-75. Interface to Different Port-Size Devices**

Table 13-43 lists the bytes required on the data bus for read cycles.

**Table 13-43. Data Bus Drive Requirements For Read Cycles**

| Transfer Size | Address State [1] 3 lsbs | Port Size/LAD Data Bus Assignments | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 32-Bit | | | | 16-Bit | | | | 8-Bit | | | |
| | | 0–7 | 8–15 | 16–23 | 24–31 | 0–7 | 8–15 | 16–23 | 24–31 | 0–7 | 8–15 | 16–23 | 24–31 |
| Byte | 000 | OP0[2] | —[3] | — | — | OP0 | — | | | OP0 | | | |
| | 001 | — | OP1 | — | — | — | OP1 | | | OP1 | | | |
| | 010 | — | — | OP2 | — | OP2 | — | | | OP2 | | | |
| | 011 | — | — | — | OP3 | — | OP3 | | | OP3 | | | |
| | 100 | OP4 | — | — | — | OP4 | — | | | OP4 | | | |
| | 101 | — | OP5 | — | — | — | OP5 | | | OP5 | | | |
| | 110 | — | — | OP6 | — | OP6 | — | | | OP6 | | | |
| | 111 | — | — | — | OP7 | — | OP7 | | | OP7 | | | |

**Table 13-43. Data Bus Drive Requirements For Read Cycles (continued)**

| Transfer Size | Address State [1] 3 lsbs | Port Size/LAD Data Bus Assignments | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 32-Bit | | | | 16-Bit | | | | 8-Bit | | | |
| | | 0–7 | 8–15 | 16–23 | 24–31 | 0–7 | 8–15 | 16–23 | 24–31 | 0–7 | 8–15 | 16–23 | 24–31 |
| Half Word | 000 | OP0 | OP1 | — | — | OP0 | OP1 | | | OP0 | | | |
| | 001 | — | OP1 | OP2 | — | — | OP1 | | | OP1 | | | |
| | 010 | — | — | OP2 | OP3 | OP2 | OP3 | | | OP2 | | | |
| | 100 | OP4 | OP5 | — | — | OP4 | OP5 | | | OP4 | | | |
| | 101 | — | OP5 | OP6 | — | — | OP5 | | | OP5 | | | |
| | 110 | — | — | OP6 | OP7 | OP6 | OP7 | | | OP6 | | | |
| Word | 000 | OP0 | OP1 | OP2 | OP3 | OP0 | OP1 | | | OP0 | | | |
| | 100 | OP4 | OP5 | OP6 | OP7 | OP4 | OP5 | | | OP4 | | | |

[1] Address state is the calculated address for port size.

[2] OP*n*: These lanes are read or written during that bus transaction. OP0 is the most-significant byte of a doubleword operand and OP7 is the least-significant byte.

[3] — Denotes a byte not driven during that read cycle.

## 13.5.4 Command Sequence Examples for NAND Flash EEPROM

In order to program the eLBC and FCM for executing NAND Flash command sequences, command codes and pause states should be obtained from the relevant NAND Flash device data sheet and programmed into FCM configuration registers. This section illustrates some common sequences for large-page, multi-gigabit NAND Flash EEPROMs; however, details should be verified against manufacturers' specific programming data.

Throughout these examples it is assumed that one or more banks of eLBC has been configured under FCM control (BR*n*[MSEL] = 001), with base address, port size, ECC mode, and timing parameters configured in accordance with the device's hardware specifications.

### 13.5.4.1 NAND Flash Soft Reset Command Sequence Example

An example of configuring FCM to execute a soft reset command to large-page NAND Flash is shown in Table 13-44. This sequence does not require use of the shared FCM buffer RAM. The sequence is initiated by writing FMR[OP] = 10, and issuing a special operation to the bank. At the conclusion of the sequence, eLBC will issue a command complete interrupt (LTESR[CC]) if interrupts are enabled.

**Table 13-44. FCM Register Settings for Soft Reset (OR*n*[PGS] = 1)**

| Register | Initial Contents | Description |
|---|---|---|
| FCR | 0xFF000000 | CMD0 = 0xFF = reset command; other commands unused |
| FBAR | — | unused |

Table 13-44. FCM Register Settings for Soft Reset (OR*n*[PGS] = 1) (continued)

| Register | Initial Contents | Description |
|---|---|---|
| FPAR | — | unused |
| FBCR | — | unused |
| MDR | — | unused |
| FIR | 0x40000000 | OP0 = CM0 = command 0;<br>OP1–OP7 = NOP |

### 13.5.4.2 NAND Flash Read Status Command Sequence Example

An example of configuring FCM to execute a status read command to large-page NAND Flash is shown in Table 13-45. This sequence does not require use of the shared FCM buffer RAM, but reads the NAND Flash status into register MDR[AS0]. The sequence is initiated by writing FMR[OP] = 10 and issuing a special operation to the bank. At the conclusion of the sequence, eLBC will issue a command complete interrupt (LTESR[CC]) if interrupts are enabled.

Table 13-45. FCM Register Settings for Status Read (OR*n*[PGS] = 1)

| Register | Initial Contents | Description |
|---|---|---|
| FCR | 0x70000000 | CMD0 = 0x70 = read status command; other commands unused |
| FBAR | — | unused |
| FPAR | — | unused |
| FBCR | — | unused |
| MDR | — | Status returned in AS0 |
| FIR | 0x4B000000 | OP0 = CM0 = command 0;<br>OP1 = RS = read status to MDR;<br>OP2–OP7 = NOP |

### 13.5.4.3 NAND Flash Read Identification Command Sequence Example

An example of configuring FCM to execute a status ID command to large-page NAND Flash is shown in Table 13-46. This sequence does not require use of the shared FCM buffer RAM, but uses MDR to set up a dummy address prior to the sequence, and then to receive the first 4 bytes of ID during the sequence. The sequence is initiated by writing FMR[OP] = 10, and issuing a special operation to the bank. At the conclusion of the sequence, eLBC will issue a command complete interrupt (LTESR[CC]) if interrupts are enabled. MDR[AS3–AS0] then can be read to obtain the first 4 bytes of NAND Flash ID.

Table 13-46. FCM Register Settings for ID Read (OR*n*[PGS] = 1)

| Register | Initial Contents | Description |
|---|---|---|
| FCR | 0x90000000 | CMD0 = 0x90 = read ID command; other commands unused |
| FBAR | — | unused |
| FPAR | — | unused |

**Table 13-46. FCM Register Settings for ID Read (OR*n*[PGS] = 1) (continued)**

| Register | Initial Contents | Description |
|---|---|---|
| FBCR | — | unused |
| MDR | 0x00000000 | AS0 = 0x00 = dummy address for read ID command;<br>AS0–AS3 return with first 4 bytes of ID code |
| FIR | 0x43BBBBB0 | OP0 = CM0 = command 0;<br>OP1 = UA = user address from MDR;<br>OP2–OP6 = RS = read 4 bytes ID into MDR[AS3–AS0];<br>OP7 = NOP |

### 13.5.4.4 NAND Flash Page Read Command Sequence Example

An example of configuring FCM to execute a random page read command to large-page NAND Flash is shown in Table 13-47. This sequence reads an entire page (main and spare region) into the shared FCM buffer RAM, checking ECC as it proceeds. The sequence is initiated by writing FMR[OP] = 11, and issuing a special operation to the bank. A few cycles before completion itself, FECCn gets updated with the ECC bytes for the main region validated by FECCn[0]. At the conclusion of the sequence, eLBC will issue a command complete interrupt (LTESR[CC]) if interrupts are enabled. Once the sequence has completed, the shared buffer (buffer 1 for page index 5) and transfer error registers (LTECCR that reports the 512 blocks with unibit /multibit errors if any) are valid.

**Table 13-47. FCM Register Settings for Page Read (OR*n*[PGS] = 1)**

| Register | Initial Contents | Description |
|---|---|---|
| FCR | 0x00300000 | CMD0 = 0x00 = random read address entry;<br>CMD1 = 0x30 = read page |
| FBAR | block index<br>(e.g. block 0x00010ab4) | BLK locates index of 128-Kbyte block |
| FPAR | page offset<br>(e.g. 0x00005000<br>locates page 5, buffer 1) | PI locates page index in BLK;<br>PI mod 2 indexes FCM buffer RAM;<br>MS = 0 and CI = 0 |
| FBCR | 0x00000000 | BC = 0 to read entire 2112-byte page with ECC check |
| MDR | — | unused |
| FIR | 0x4125E000 | OP0 = CM0 = command 0;<br>OP1 = CA = column address;<br>OP2 = PA = page address;<br>OP3 = CM1 = command 1;<br>OP4 = RBW = wait on Flash ready and read data into FCM buffer;<br>OP5–OP7 = NOP |

### 13.5.4.5 NAND Flash Block Erase Command Sequence Example

An example of configuring FCM to execute a block erase command to large-page NAND Flash is shown in Table 13-48. This sequence does not require use of the shared FCM buffer RAM, but returns with the erase status in MDR[AS0]. The sequence is initiated by writing FMR[OP] = 11, and issuing a special operation to the bank. At the conclusion of the sequence, eLBC will issue a command complete interrupt (LTESR[CC]) if interrupts are enabled.

Note that operations specified by OP3 and OP4 (status read) should never be skipped while erasing a NAND Flash device, because, in case that happens, contention may arise on LGPL4. A possible case is that the next transaction from eLBC may try to use that pin as an output and since the NAND Flash device might already be driving it, contention will occur. In case OP3 and OP4 operations are skipped, it may also happen that a new command is issued to the NAND Flash device even when the device has not yet finished processing the previous request. This may also result in unpredictable behavior.

**Table 13-48. FCM Register Settings for Block Erase (OR*n*[PGS] = 1)**

| Register | Initial Contents | Description |
|---|---|---|
| FCR | 0x6070D000 | CMD0 = 0x60 = block address entry;<br>CMD1 = 0x70 = read status<br>CMD2 = 0xD0 = erase block; |
| FBAR | block index<br>(e.g. block 0x00010AB4) | BLK locates index of 128-Kbyte block |
| FPAR | 0x00000000 | PI = 0 to locate block boundary |
| FBCR | — | unused |
| MDR | — | returns with AS0 holding erase status |
| FIR | 0x426DB000 | OP0 = CM0 = command 0;<br>OP1 = PA = page address;<br>OP2 = CM2 = command 2;<br>OP3 = CW1 = wait on Flash ready and issue command 1;<br>OP4 = RS = read erase status into MDR[AS0];<br>OP5–OP7 = NOP |

### 13.5.4.6 NAND Flash Program Command Sequence Example

An example of configuring FCM to execute a program command to large-page NAND Flash is shown in Table 13-49. This sequence writes an entire page (main and spare region) from the shared FCM buffer RAM, generating ECC as it proceeds. The shared buffer (buffer 1 for page index 5) must be initialized by software prior to starting the sequence. The sequence is initiated by writing FMR[OP] = 11, and issuing a special operation to the bank. At the conclusion of the sequence, eLBC will issue a command complete interrupt (LTESR[CC]) if interrupts are enabled. The status of the programming operation is returned in MDR[AS0].

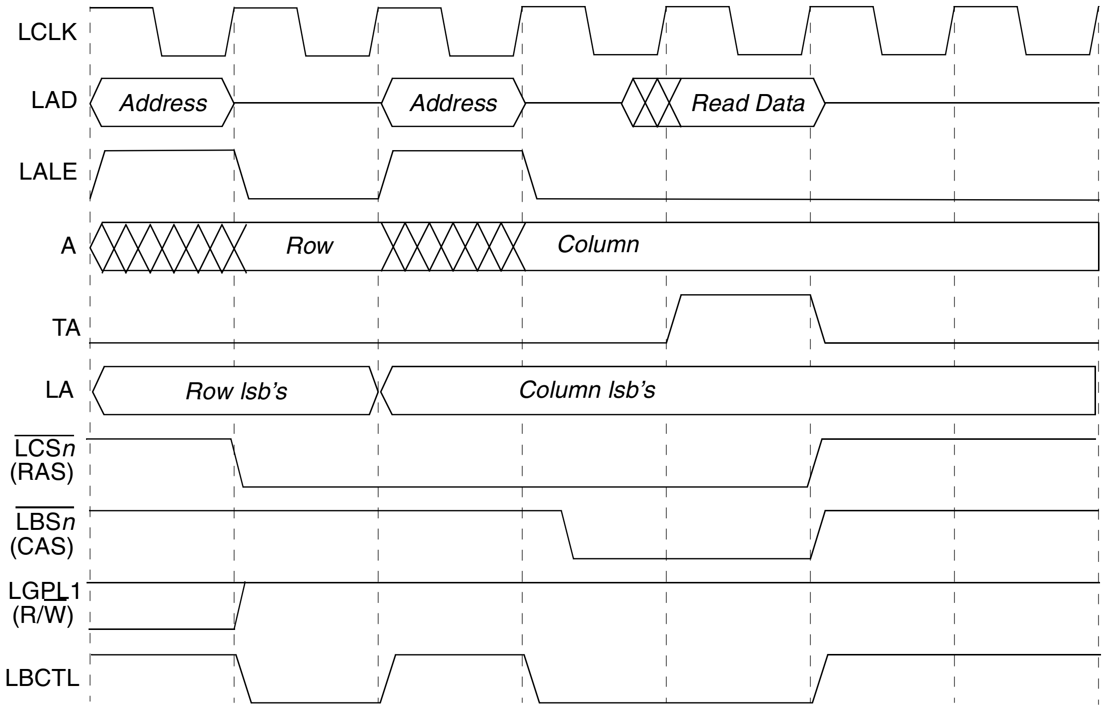
Note that operations specified by OP5 and OP6 (status read) should never be skipped while programming a NAND Flash device, because, in case that happens, contention may arise on LGPL4. A possible case is that the next transaction from eLBC may try to use that pin as an output and since the NAND Flash device might already be driving it, contention will occur. In case OP5 and OP6 operations are skipped, it may also happen that a new command is issued to the NAND Flash device even when the device has not yet finished processing the previous request. This may also result in unpredictable behavior.

**Table 13-49. FCM Register Settings for Page Program (OR*n*[PGS] = 1)**

| Register | Initial Contents | Description |
|---|---|---|
| FCR | 0x80701000 | CMD0 = 0x80 = page address and data entry;<br>CMD1 = 0x70 = read status<br>CMD2 = 0x10 = program page; |
| FBAR | block index<br>(e.g. block 0x00010AB4) | BLK locates index of 128-Kbyte block |
| FPAR | page offset<br>(e.g. 0x00005000<br>locates page 5, buffer 1) | PI locates page index in BLK;<br>PI mod 2 indexes FCM buffer RAM;<br>MS = 0 and CI = 0 |
| FBCR | 0x00000000 | BC = 0 to write entire 2112-Byte page with ECC generation |
| MDR | — | returns with AS0 holding program status |
| FIR | 0x41286DB0 | OP0 = CM0 = command 0;<br>OP1 = CA = column address;<br>OP2 = PA = page address;<br>OP3 = WB = write data from buffer;<br>OP4 = CM2 = command 2;<br>OP5 = CW1 = wait on Flash ready and issue command 1;<br>OP6 = RS = read erase status into MDR[AS0];<br>OP7 = NOP |

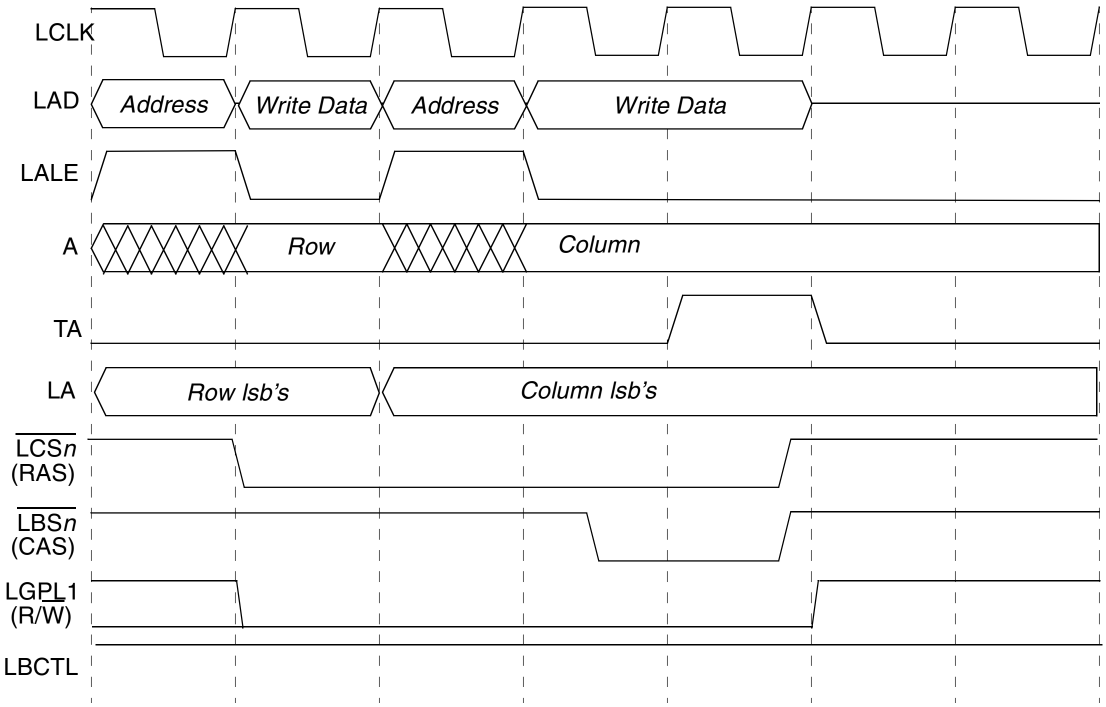
### 13.5.5 Interfacing to Fast-Page Mode DRAM Using UPM

Connecting the local bus UPM controller to a DRAM device requires a detailed examination of the timing diagrams representing the possible memory cycles that must be performed when accessing this device. This section describes timing diagrams for various UPM configurations for fast-page mode DRAM, with LCRR[CLKDIV] = 4 (clock ratio of 8) or 8 (clock ratio of 16). These illustrative examples may not represent the timing necessary for any specific device used with the eLBC. Here, LGPL1 is programmed to drive R/$\overline{W}$ of the DRAM, although any LGPL*n* signal may be used for this purpose.



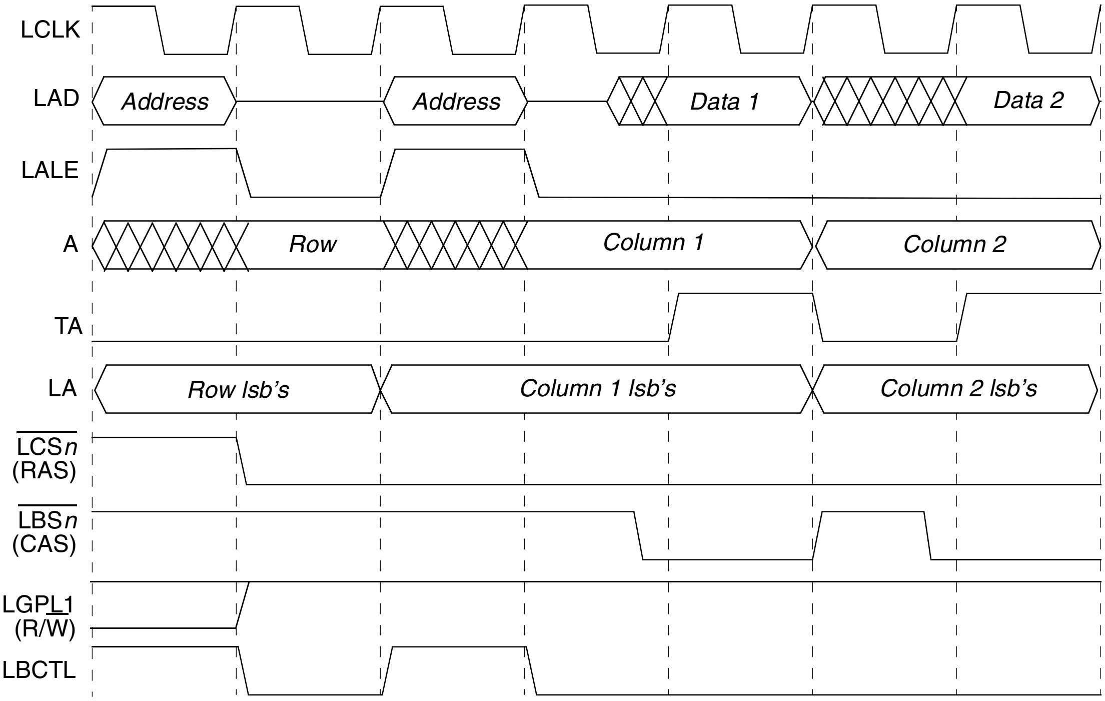
| | | | | | |
|---|---|---|---|---|---|
| cst1 | 0 | LALE pause (due to change in AMX) | 0 | 0 | Bit 0 |
| cst2 | 0 | | 0 | 0 | Bit 1 |
| cst3 | 0 | | 0 | 0 | Bit 2 |
| cst4 | 0 | | 0 | 0 | Bit 3 |
| bst1 | 1 | | 1 | 0 | Bit 4 |
| bst2 | 1 | | 0 | 0 | Bit 5 |
| bst3 | 1 | | 0 | 0 | Bit 6 |
| bst4 | 1 | | 0 | 0 | Bit 7 |
| g0l0 | | | | | Bit 8 |
| g0l1 | | | | | Bit 9 |
| g0h0 | | | | | Bit 10 |
| g0h1 | | | | | Bit 11 |
| g1t1 | 1 | | 1 | 1 | Bit 12 |
| g1t3 | 1 | | 1 | 1 | Bit 13 |
| g2t1 | | | | | Bit 14 |
| g2t3 | | | | | Bit 15 |
| g3t1 | | | | | Bit 16 |
| g3t3 | | | | | Bit 17 |
| g4t1 | | | | | Bit 18 |
| g4t3 | | | | | Bit 19 |
| g5t1 | | | | | Bit 20 |
| g5t3 | | | | | Bit 21 |
| redo[0] | | | | | Bit 22 |
| redo[1] | | | | | Bit 23 |
| loop | 0 | | 0 | 0 | Bit 24 |
| exen | 0 | | 0 | 0 | Bit 25 |
| amx0 | 1 | | 0 | 0 | Bit 26 |
| amx1 | 0 | | 0 | 0 | Bit 27 |
| na | 0 | | 0 | 0 | Bit 28 |
| uta | 0 | | 0 | 1 | Bit 29 |
| todt | 0 | | 0 | 1 | Bit 30 |
| last | 0 | | 0 | 1 | Bit 31 |
| | **RSS** | **RSS+1** | **RSS+1** | **RSS+2** | |

**Figure 13-76. Single-Beat Read Access to FPM DRAM**



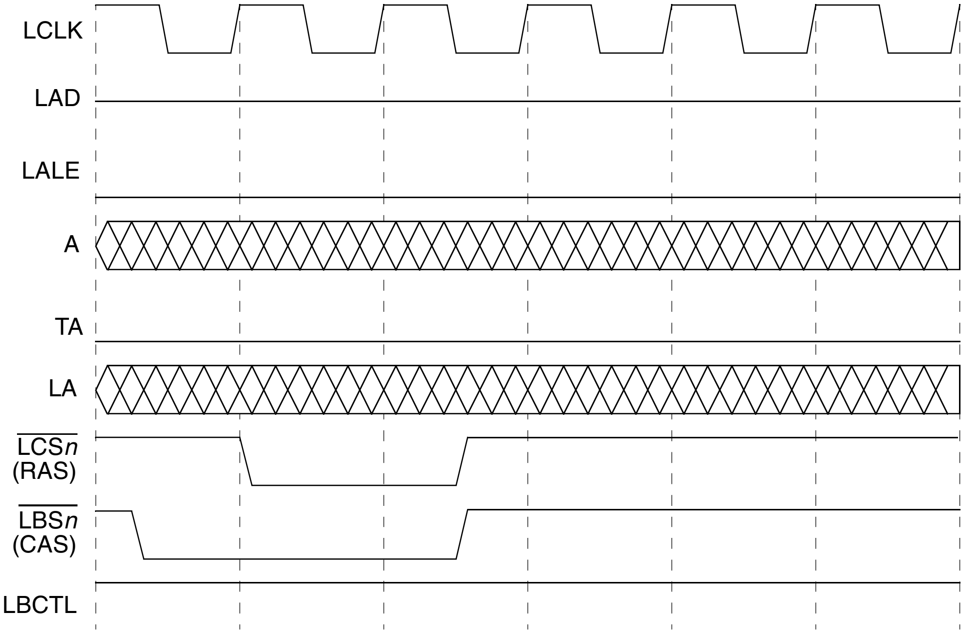
| | | | | | |
|---|---|---|---|---|---|
| cst1 | 0 | LALE pause (due to change in AMX) | 0 | 0 | Bit 0 |
| cst2 | 0 | | 0 | 0 | Bit 1 |
| cst3 | 0 | | 0 | 0 | Bit 2 |
| cst4 | 0 | | 0 | 1 | Bit 3 |
| bst1 | 1 | | 1 | 0 | Bit 4 |
| bst2 | 1 | | 1 | 0 | Bit 5 |
| bst3 | 1 | | 0 | 0 | Bit 6 |
| bst4 | 1 | | 0 | 1 | Bit 7 |
| g0l0 | | | | | Bit 8 |
| g0l1 | | | | | Bit 9 |
| g0h0 | | | | | Bit 10 |
| g0h1 | | | | | Bit 11 |
| g1t1 | 0 | | 0 | 0 | Bit 12 |
| g1t3 | 0 | | 0 | 0 | Bit 13 |
| g2t1 | | | | | Bit 14 |
| g2t3 | | | | | Bit 15 |
| g3t1 | | | | | Bit 16 |
| g3t3 | | | | | Bit 17 |
| g4t1 | | | | | Bit 18 |
| g4t3 | | | | | Bit 19 |
| g5t1 | | | | | Bit 20 |
| g5t3 | | | | | Bit 21 |
| redo[0] | | | | | Bit 22 |
| redo[1] | | | | | Bit 23 |
| loop | 0 | | 0 | 0 | Bit 24 |
| exen | 0 | | 0 | 0 | Bit 25 |
| amx0 | 1 | | 0 | 0 | Bit 26 |
| amx1 | 0 | | 0 | 0 | Bit 27 |
| na | 0 | | 0 | 0 | Bit 28 |
| uta | 0 | | 0 | 1 | Bit 29 |
| todt | 0 | | 0 | 1 | Bit 30 |
| last | 0 | | 0 | 1 | Bit 31 |
| | **WSS** | **WSS+1** | **WSS+1** | **WSS+2** | |

**Figure 13-77. Single-Beat Write Access to FPM DRAM**



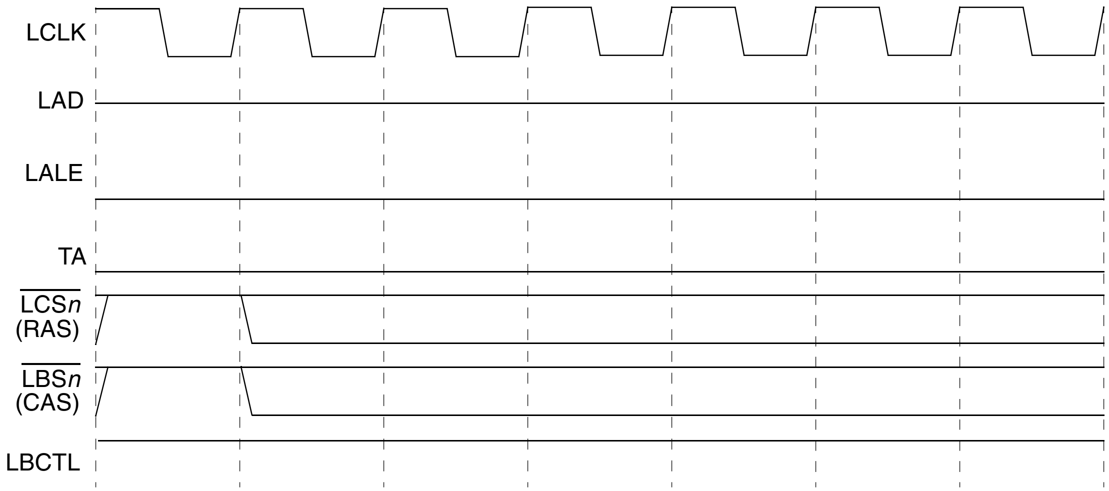
| | | | | | | |
|---|---|---|---|---|---|---|
| cst1 | 0 | LALE pause (due to change in AMX) | 0 | 0 | 1 | Bit 0 |
| cst2 | 0 | | 0 | 0 | 1 | Bit 1 |
| cst3 | 0 | | 0 | 0 | 1 | Bit 2 |
| cst4 | 0 | | 0 | 0 | 1 | Bit 3 |
| bst1 | 1 | | 1 | 0 | 1 | Bit 4 |
| bst2 | 1 | | 1 | 0 | 1 | Bit 5 |
| bst3 | 1 | | 1 | 0 | 1 | Bit 6 |
| bst4 | 1 | | 0 | 0 | 1 | Bit 7 |
| g0l0 | | | | | | Bit 8 |
| g0l1 | | | | | | Bit 9 |
| g0h0 | | | | | | Bit 10 |
| g0h1 | | | | | | Bit 11 |
| g1t1 | 1 | | 1 | 1 | 1 | Bit 12 |
| g1t3 | 1 | | 1 | 1 | 1 | Bit 13 |
| g2t1 | | | | | | Bit 14 |
| g2t3 | | | | | | Bit 15 |
| g3t1 | | | | | | Bit 16 |
| g3t3 | | | | | | Bit 17 |
| g4t1 | | | | | | Bit 18 |
| g4t3 | | | | | | Bit 19 |
| g5t1 | | | | | | Bit 20 |
| g5t3 | | | | | | Bit 21 |
| redo[0] | | | | | | Bit 22 |
| redo[1] | | | | | | Bit 23 |
| loop | 0 | | 1 | 1 | 0 | Bit 24 |
| exen | 0 | | 0 | 1 | 0 | Bit 25 |
| amx0 | 1 | | 0 | 0 | 0 | Bit 26 |
| amx1 | 0 | | 0 | 0 | 0 | Bit 27 |
| na | 0 | | 0 | 1 | 0 | Bit 28 |
| uta | 0 | | 0 | 1 | 0 | Bit 29 |
| todt | 0 | | 0 | 0 | 1 | Bit 30 |
| last | 0 | | 0 | 0 | 1 | Bit 31 |
| | **RBS** | | **RBS+1** | **RBS+2** | **RBS+3** | |

**Figure 13-78. Burst Read Access to FPM DRAM Using LOOP (Two Beats Shown)**


LCLK
LAD
LALE
A
TA
LA
$\overline{\text{LCS}n}$ (RAS)
$\overline{\text{LBS}n}$ (CAS)
LBCTL

| | PTS | PTS+1 | PTS+2 | |
|---|---|---|---|---|
| cst1 | 1 | 0 | 0 | Bit 0 |
| cst2 | 1 | 0 | 0 | Bit 1 |
| cst3 | 1 | 0 | 1 | Bit 2 |
| cst4 | 1 | 0 | 1 | Bit 3 |
| bst1 | 1 | 0 | 0 | Bit 4 |
| bst2 | 0 | 0 | 0 | Bit 5 |
| bst3 | 0 | 0 | 1 | Bit 6 |
| bst4 | 0 | 0 | 1 | Bit 7 |
| g0l0 | | | | Bit 8 |
| g0l1 | | | | Bit 9 |
| g0h0 | | | | Bit 10 |
| g0h1 | | | | Bit 11 |
| g1t1 | | | | Bit 12 |
| g1t3 | | | | Bit 13 |
| g2t1 | | | | Bit 14 |
| g2t3 | | | | Bit 15 |
| g3t1 | | | | Bit 16 |
| g3t3 | | | | Bit 17 |
| g4t1 | | | | Bit 18 |
| g4t3 | | | | Bit 19 |
| g5t1 | | | | Bit 20 |
| g5t3 | | | | Bit 21 |
| redo[0] | | | | Bit 22 |
| redo[1] | | | | Bit 23 |
| loop | 0 | 0 | 0 | Bit 24 |
| exen | 0 | 0 | 0 | Bit 25 |
| amx0 | 0 | 0 | 0 | Bit 26 |
| amx1 | 0 | 0 | 0 | Bit 27 |
| na | 0 | 0 | 0 | Bit 28 |
| uta | 0 | 0 | 0 | Bit 29 |
| todt | 0 | 0 | 1 | Bit 30 |
| last | 0 | 0 | 1 | Bit 31 |

**Figure 13-79. Refresh Cycle (CBR) to FPM DRAM**



| | | |
|---|---|---|
| cst1 | 1 | Bit 0 |
| cst2 | 1 | Bit 1 |
| cst3 | 1 | Bit 2 |
| cst4 | 1 | Bit 3 |
| bst1 | 1 | Bit 4 |
| bst2 | 1 | Bit 5 |
| bst3 | 1 | Bit 6 |
| bst4 | 1 | Bit 7 |
| g0l0 | | Bit 8 |
| g0l1 | | Bit 9 |
| g0h0 | | Bit 10 |
| g0h1 | | Bit 11 |
| g1t1 | | Bit 12 |
| g1t3 | | Bit 13 |
| g2t1 | | Bit 14 |
| g2t3 | | Bit 15 |
| g3t1 | | Bit 16 |
| g3t3 | | Bit 17 |
| g4t1 | | Bit 18 |
| g4t3 | | Bit 19 |
| g5t1 | | Bit 20 |
| g5t3 | | Bit 21 |
| redo[0] | | Bit 22 |
| redo[1] | | Bit 23 |
| loop | 0 | Bit 24 |
| exen | 0 | Bit 25 |
| amx0 | 0 | Bit 26 |
| amx1 | 0 | Bit 27 |
| na | 0 | Bit 28 |
| uta | 0 | Bit 29 |
| todt | 1 | Bit 30 |
| last | 1 | Bit 31 |
| | **EXS** | |

**Figure 13-80. Exception Cycle**

## 13.5.6 Interfacing to ZBT SRAM Using UPM

ZBT SRAMs have been designed to optimize the performance of table access in networking applications. This section describes how to interface to ZBT SRAMs. Figure 13-81 shows the connections. The UPM is used to generate control signals. The same interfacing is used for pipelined and flow-through versions of ZBT SRAMs. However different UPM patterns must be generated for those cases. Because ZBT

SRAMs will mostly be used by performance-critical applications, we assume here that, typically, the maximum width of the local bus of 32 bits will be used.

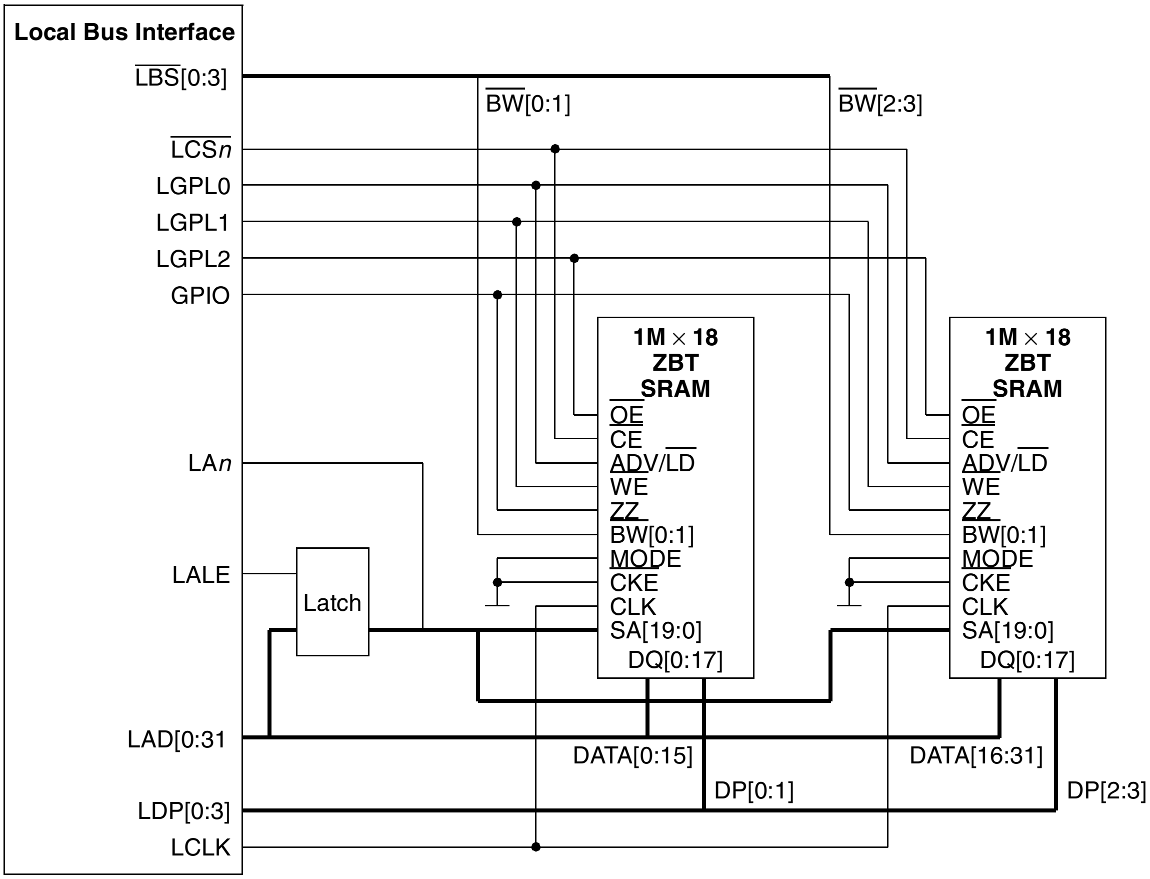


**Figure 13-81. Interface to ZBT SRAM**

ZBT SRAMs allow different configurations. For the local bus, the burst order should be set to linear burst order by tying the mode pin to GND. CKE should also be tied to ground.

ZBT SRAMs perform four-beat bursts. Because the eLBC generates eight-beat transactions (for 32-bit ports) the UPM breaks down each burst into two consecutive four-beat bursts. The internal address generator of the eLBC generates the new LA bits for each burst. In other words, because linear burst is used on the SRAM, the device itself bursts with the burst addresses of [0:1:2:3]. The local bus always generates linear bursts and expects [0:1:2:3:4:5:6:7]. Therefore, two consecutive linear bursts of the ZBT SRAM with {A27, A28} = {0,0} for the first burst, and {A27, A28} = {1,0} for the second burst give the desired burst pattern.

The UPM also supports single beat accesses. Because the ZBT SRAM does not support this and always responds with a burst, the UPM pattern has to take care that data for the critical beat is provided (for write) or sampled (for read), and that the rest of the burst is ignored (by negating WE). The UPM controller basically has to wait for the end of the SRAM burst to avoid bus contention with further bus activities.

# Chapter 14
# Enhanced Three-Speed Ethernet Controllers

## 14.1 Overview

The enhanced three-speed Ethernet controllers (eTSECs) of the device interface to 10 Mbps, 100 Mbps, and 1 Gbps Ethernet/IEEE 802.3™ networks and devices featuring generic 8-bit FIFO ports. For Ethernet, an external PHY or SerDes device is required to complete the interface to the media. Each eTSEC supports multiple standard media-independent interfaces, of which the FIFO interface bypasses the Ethernet MAC. Multiple eTSECs are available, providing flexible options for connectivity and control access at different speeds.

The eTSEC provides the flexibility to accelerate the identification and retrieval of standard and non-standard protocols carried over Ethernet, including both IP versions 4 and 6 and TCP/UDP. CPU-intensive parsing and checksum operations can be optionally off-loaded to an eTSEC to accelerate existing TCP/IP stacks. On transmission, varying fractions of link bandwidth can be allocated to each of multiple transmit queues through a modified weighted round-robin scheduler. On receive, an arbitrary set of queue selection rules can be programmed into each eTSEC to implement flexible quality of service or firewall strategies based on high-level protocol identification. Without enabling these advanced features, each eTSEC emulates a PowerQUICC  TSEC, allowing existing driver software to be re-used with minimal change. Each eTSEC is organized as shown in Figure 14-1.

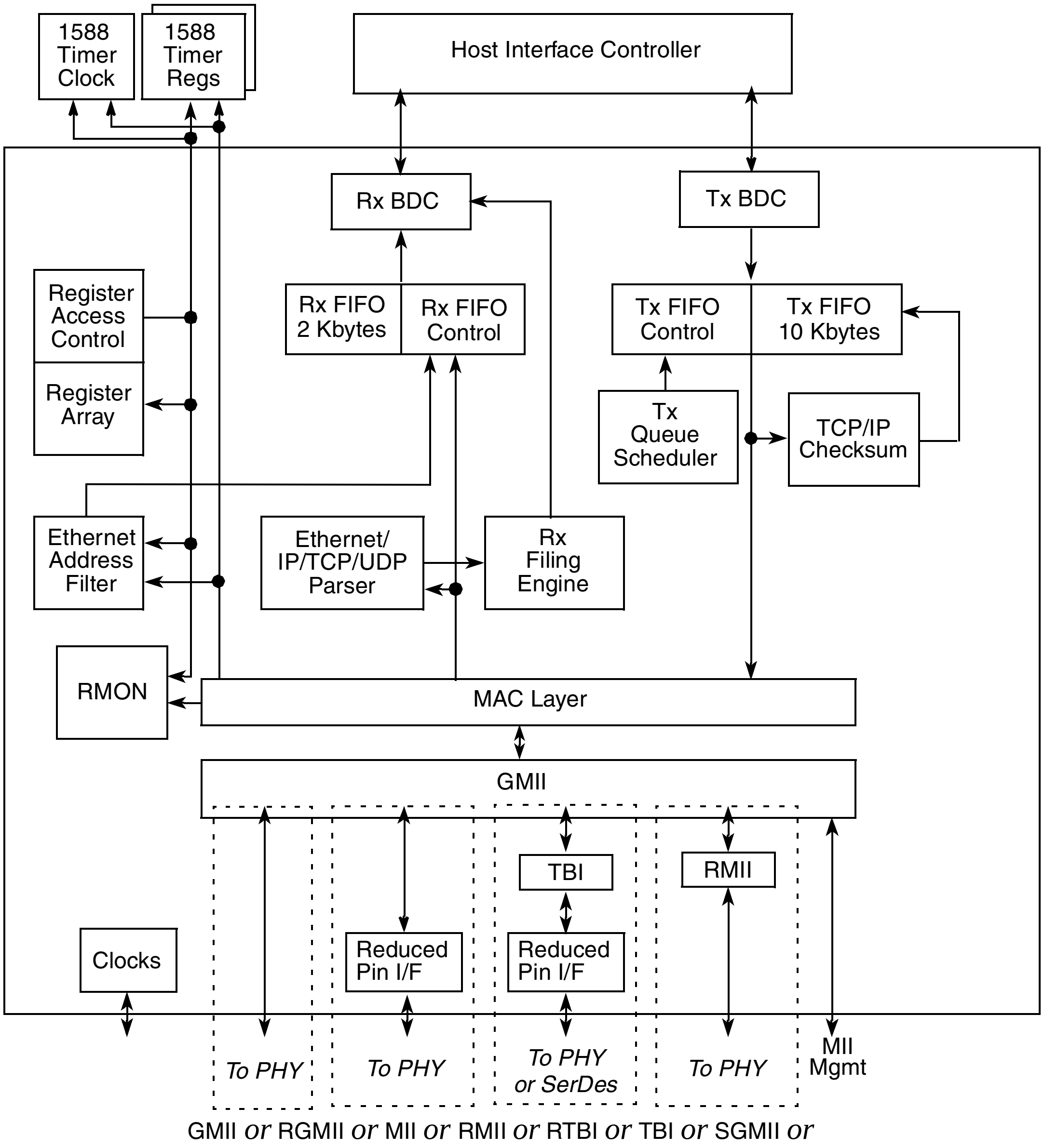


**Figure 14-1. eTSEC Block Diagram**

## 14.2 Features

The eTSECs of the device include these distinctive features:

- IEEE 802.3, 802.3u, 820.3x, 802.3z, 802.3ac, 802.3ab compatible
- Support for different Ethernet physical interfaces:
  - — 10/100 Mbps IEEE 802.3 GMII
  - — 1000 Mbps full-duplex IEEE 802.3 GMII
  - — 10/100 Mbps IEEE 802.3 MII and RMII

- — 10/100 Mbps RGMII
- — 1000 Mbps full-duplex RGMII and RTBI
- — 10/100 Mbps SGMII
- — 1000 Mbps full-duplex SGMII
- — 1000 Mbps IEEE 802.3z TBI
- — Single-clock TBI

- Support for two full-duplex FIFO interface modes
  - — 8-bit mode—GMII style and encoded packet
  - — Inter-packet and intra-packet flow control
  - — Optional CRC-32 generation and checking
  - — Minimal glue logic required to support POS PHY Level 3 conversion
  - — TCP/IP off-load and QoS features available in all FIFO modes
- TCP/IP off-load
  - — IP v4 and IP v6 header recognition on receive
  - — IP v4 header checksum verification and generation
  - — TCP and UDP checksum verification and generation
  - — Per-packet configurable off-load
  - — Recognition of VLAN, stacked-VLAN, 802.2, PPPoE session, MPLS stacks, ARP, and ESP/AH IP-Security headers
- Quality of service (QoS) support
  - — Transmission from up to eight queues
    - – Priority-based queue selection
    - – Modified weighted round-robin queue selection with fair bandwidth allocation
  - — Reception to up to eight physical queues
    - – 64 virtual receive queues overlaid on 8 physical buffer descriptor rings
    - – Table-oriented queue filing strategy based on 16 header fields or flags
    - – Frame rejection support for filtering applications
    - – Filing based on Ethernet, IP, and TCP/UDP properties, including VLAN fields, Ether-type, IP protocol type, IP TOS or differentiated services, IP source and destination addresses, TCP/UDP port numbers
- Interrupt coalescing
  - — Packet-count-based thresholds for both receive and transmit
  - — Timer-based thresholds
- Full- and half-duplex Ethernet support (1000 Mbps supports only full duplex):
  - — IEEE 802.3 full-duplex flow control (automatic PAUSE frame generation or software programmed PAUSE frame generation and recognition)
  - — Programmable maximum frame length supports jumbo frames (up to 9.6 Kbytes) and IEEE 802.1 virtual local area network (VLAN) tags and priority

  - VLAN insertion and deletion
    - Per-frame VLAN control word or default VLAN for each eTSEC
    - Extracted VLAN control word passed to software separately
    - Programmable VLAN tag to support metropolitan bridging
  - Retransmission following a collision
  - Support for CRC generation and verification of inbound/outbound packets
  - Programmable Ethernet preamble insertion and extraction of up to 7 bytes
- MAC address recognition:
  - Exact match on primary and virtual 48-bit unicast addresses
    - VRRP and HSRP support for seamless router fail-over
    - In addition to primary station address, up to fifteen additional exact-match MAC addresses supported
  - Broadcast address (accept/reject)
  - Hash table match on up to 256 unicast/multicast or 512 multicast-only addresses
  - Promiscuous mode
- Remote network monitoring (RMON) statistics support
  - 32-bit byte counters
  - Carry/Overflow of counter interrupts
- Backward compatibility with MPC8540E/MPC8560E (PowerQUICC III) TSEC
  - PowerQUICC III buffer descriptor (BD) format and rings supported
  - Common register memory map, with specific exceptions:
    - Out-of-sequence transmit BD not supported
    - Internal DMA BD pointers and data counts not visible
    - MINFLR register not supported
  - Reset state of eTSEC defaults to common PowerQUICC III TSEC subset
  - TSEC_ID register permits TSEC versus enhanced TSEC differentiation
- Hardware assist for 1588 compliant timestamping (1588 not supported in conjunction with SGMII 10/100)
  - Per packet timestamp tag for Receive
  - Programmable timestamp capture for Transmit
  - Recognition of PTP packet
  - Periodic Pulse Generation
  - Self-correcting precision timer with nano-second resolution
  - Phase aligned adjustable (divide by N) clock output
  - Two 64-bit alarm (future time) registers for future time comparison

## 14.3 Modes of Operation

The eTSEC's primary operational modes are the following:

- Ethernet and FIFO operation

  The ECNTRL register's FIFO mode enable bit (ECNTRL[FIFM]) allows bypass of the Ethernet MAC and enables I/O through the FIFO interface sharing the normal GMII signals. Each eTSEC supports an 8-bit FIFO interface independently. If configured in FIFO mode, the FIFOCFG register determines operation. In FIFO mode data is transferred synchronously with respect to the external data clock. See the device hardware specifications document for maximum supported frequencies.

- Full- and half-duplex operation

  This is determined by the MACCFG2 register's full-duplex bit (MACCFG2[Full Duplex]). Full-duplex mode is intended for use on point-to-point links between switches or end node to switch. Half-duplex mode is used in connections between an end node and a repeater or between repeaters.

  If configured in half-duplex mode (10- and 100-Mbps operation; MACCFG2[Full Duplex] is cleared), the MAC complies with the IEEE CSMA/CD access method.

  If configured in full-duplex mode (10/100/1000 Mbps operation; MACCFG2[Full Duplex] is set), the MAC supports flow control. If flow control is enabled, it allows the MAC to receive or send PAUSE frames.

- 10- and 100-Mbps MII interface operation

  The MAC–PHY interface operates in MII mode by setting MACCFG2[I/F Mode] = 01. The MII is the media-independent interface defined by the 802.3 standard for 10/100 Mbps operation. The speed of operation is determined by the TSEC*n*_TX_CLK and TSEC*n*_RX_CLK signals, which are driven by the transceiver. The transceiver either auto-negotiates the speed, or it may be controlled by software using the serial management interface (MDC/MDIO signals) to the transceiver.

  Clause 22.2.4 of the IEEE 802.3 specification describes the MII management interface.

- 10- and 100-Mbps RMII interface operation

  The RMII is the reduced media-independent interface defined by the RMII Consortium (March 1998) for 10/100 Mbps operation. The speed of operation is determined by the TSEC*n*_TX_CLK signal, which is driven by the transceiver.

- 1000 Mbps GMII and TBI interface operation

  The MAC–PHY interface operates in GMII mode by setting MACCFG2[I/F Mode] = 10. The GMII is the gigabit media-independent interface defined by the 802.3 standard for 1000-Mbps operation.

  Independently, the MAC-PHY interface can also operate in TBI mode. Note that either the TBI or GMII interface is chosen, not both at the same time. TBI is the 10-bit interface which contains PCS functions (10-bit encoding/decoding) as defined by the 802.3 standard.

  In reduced-pin count mode (RGMII or RTBI), the MAC remains configured in GMII or TBI but the eTSEC muxes and decodes the input signals and provides the MAC with the expected interface.

  eTSEC provides the TSEC*n*_GTX_CLK to the PHY in either GMII or TBI mode of operation.

- MAC address recognition options

The options supported are promiscuous, broadcast, exact unicast address match, exact unicast virtual address match to support router redundancy, and multicast hash match. For detailed descriptions refer to Section 14.6.3.7, “Frame Recognition.”

eTSEC supports automatic LAN-initiated wake-up during power management through the AMD Magic Packet™ protocol, as described in Section 14.6.3.8, “Magic Packet Mode.”

- Receive frame parsing options

  Frame parsing options are to disable parsing (no TCP/IP off-load), IP header parsing, and TCP or UDP parsing. Parsing must be enabled to make use of receive queue filing algorithms. The options are detailed in Section 14.6.4, “TCP/IP Off-Load.”
- Receive queue selection options

  Received frames are by default sent to a single buffer descriptor ring. If multiple receive queues are enabled, a receive queue filer can be programmed with selection criteria to differentiate received frames and file them to different buffer descriptor rings. See Section 14.6.5, “Quality of Service (QoS) Provision,” for detailed descriptions.
- TCP/IP transmit options

  Frames for transmission may be sent as-is, with IP header processing, or TCP header processing. The transmit buffer descriptors, described in Section 14.6.8.2, “Transmit Data Buffer Descriptors (TxBD),” enable these options and operate with parameters prepended to frame buffers, as described in Section 14.6.4, “TCP/IP Off-Load.”
- Transmit queue selection options

  The options supported are single transmit queue, priority-based queue selection, and modified weighted round-robin queueing. These options are described further in Section 14.5.3.2.1, “Transmit Control Register (TCTRL).”
- RMON support

  Standard Ethernet interface management information base (MIBs) can be generated through the RMON MIB counters.
- Internal loop back supported for all interfaces except when configured for half-duplex operation

  Internal loop back mode is selected through the loop back bit in the MACCFG1 register. See Section 14.7.1, “Interface Mode Configuration,” for details.

## 14.4 External Signals Description

This section defines the eTSEC interface signals. The buses are described using the bus convention used in IEEE 802.3 because the PHY follows this same convention. (That is, TxD[7:0] means 0 is the lsb.) Note that except for external physical interfaces the buses and registers follow a big-endian format, where 0 denotes the msb.

Each eTSEC network interface supports multiple options:

- The MII option requires 18 I/O signals (including the MDIO and MDC MII management interface) and supports both a data and a management interface to the PHY (transceiver) device. The MII option supports both 10- and 100-Mbps Ethernet rates.
- The GMII option is a superset of the MII signals and supports a 1000-Mbps Ethernet rate.

- The TBI interface shares signals with the GMII interface signals.
- The RGMII, RTBI, and RMII options are reduced-pin implementations of the GMII, TBI, and MII interfaces, respectively.
- SGMII interfaces are offered via the SerDes interface signals.
- 1588 timer signals
- Finally, the FIFO interfaces share the GMII signals—8 bits of data plus 3 bits of control signals.

Table 14-1 lists the network interface signals.

**Table 14-1. eTSEC*n* Network Interface Signal Properties**

| Signal Name | Function | Reset State |
|---|---|---|
| TSEC*n*_COL | MII—collision, input<br>FIFO—transmit flow control, input | — |
| TSEC*n*_CRS | MII—carrier sense, input<br>TBI—signal detect, input<br>FIFO—receive flow control, output | — |
| TSEC*n*_GTX_CLK | RTBI, RGMII—inverted transmit clock feedback, output<br>TBIFIFO—continuous transmit clock feedback, output<br>GMII, MII, RMII—transmit clock feedback when transmission is enabled, zero otherwise, output | 0 |
| EC_GTX_CLK125 | Oscillator source for GMII, TBI, RGMII, RTBI transmit clock, input, shared by all eTSECs | — |
| EC_MDC | Management clock, output. | 0 |
| EC_MDIO | Management data, bidirectional. | Hi-Z (input) |
| TSEC*n*_RX_CLK | GMII, MII, RGMII—receive clock, input<br>TBI—PMA receive clock 0, input<br>FIFO—receive clock, input | — |
| TSEC*n*_RX_DV | GMII, MII—receive data valid, input<br>TBI—receive code group (RCG) bit 8, input<br>RGMII (RX_CLK rising)—receive data valid, input<br>RGMII (RX_CLK falling)—receive error, input<br>RTBI (RX_CLK rising)—receive code group (RCG) bit 4, input<br>RTBI (RX_CLK falling)—receive code group (RCG) bit 9, input<br>RMII—CRS_DV carrier sense/data valid, input<br>FIFO—receive data valid or receive control bit, input | — |
| TSEC*n*_RXD[7:4] | GMII—receive data bits 7:4 input<br>TBI—RCG bits 7:4, input<br>FIFO—receive data bits 7:4 input<br>MII, RGMII, RTBI, RMII—unused | — |
| TSEC*n*_RXD[3:0] | GMII, MII—Receive data bits 3:0, input<br>TBI—RGC bits 3:0, input<br>RGMII (RX_CLK rising) —Receive data bits 3:0, input<br>RGMII (RX_CLK falling)—Receive data bits 7:4, input<br>RTBI (RX_CLK rising)—RCG bits 3:0, input<br>RTBI (RX_CLK falling)—RCG bits 8:5, input<br>RMII—RXD[1:0] receive data bits, input<br>RMII—RXD[3:2] are unused<br>FIFO—Receive data bits 3:0, input | — |

### Table 14-1. eTSEC*n* Network Interface Signal Properties (continued)

| Signal Name | Function | Reset State |
|---|---|---|
| TSEC*n*_RX_ER | GMII, MII, RMII—Receive error, input<br>TBI—RGC bit 9, input<br>FIFO—Receive error or receive frame control bit, input<br>RGMII, RTBI—Unused | — |
| TSEC*n*_TX_CLK | MII—transmit clock, input<br>TBI—PMA receive clock 1, input<br>RMII—reference transmit and receive clock, input<br>FIFO—transmit clock, input<br>RGMII, RTBI—unused | — |
| TSEC*n*_TXD[7:4] | GMII—transmit data bit 7:4, output<br>TBI—transmit code group (TCG) bit 7:4, output<br>FIFO—transmit data bit 7:4, output<br>MII, RGMII, RTBI, RMII—unused | 0000 |
| TSEC*n*_TXD[3:0] | GMII, MII—Transmit data bits 3:0, output<br>TBI—TCG bits 3:0, output<br>RGMII (TX_CLK rising)—Transmit data bits 3:0, output<br>RGMII (TX_CLK falling)—Transmit data bits 7:4, output<br>RTBI (TX_CLK rising)—TCG bits 3:0, output<br>RTBI (TX_CLK falling)—TCG bits 8:5, output<br>RMII—TXD[1:0] transmit data bits, output<br>RMII—TXD[3:2] unused, output<br>FIFO—Transmit data bits 3:0, output | 0000 |
| TSEC*n*_TX_ER | GMII, MII—transmit error, output<br>RGMII, RTBI, RMII—unused, output driven zero<br>TBI—TCG bit 9, output<br>FIFO—transmit error or transmit frame control bit, output | 0 |
| TSEC*n*_TX_EN | GMII, MII, RMII—Transmit data valid, output<br>TBI—TCG bit 8, output<br>RGMII (TX_CLK rising)—Transmit data enabled, output<br>RGMII (TX_CLK falling)—Transmit error, output<br>RTBI (TX_CLK rising)—TCG bit 4, output<br>RTBI (TX_CLK falling)—TCG bit 9, output<br>FIFO—Transmit data valid or transmit control bit, output | 0 |
| TSEC_1588_CLK | 1588—Clock input<br>External high precision timer reference clock input (chip external input pin). | — |
| TSEC_1588_ CLK_OUT | 1588—Clock output<br>Phase aligned timer clock divider output (chip external output pin). | 0 |
| TSEC_1588_ TRIG_IN0 | 1588—Trigger in 0<br>External timer trigger input 0. This is an asynchronous general purpose input (chip external input pin). | — |
| TSEC_1588_ TRIG_IN1 | 1588—Trigger in 1<br>External timer trigger input 1. This is an asynchronous general purpose input (chip external input pin). | — |
| TSEC_1588_ PULSE_OUT1 | 1588—Pulse out 1<br>Timer pulse per period 1. It is phase aligned with 1588 timer clock (chip external output pin). | 0 |
| TSEC_1588_ PULSE_OUT2 | 1588—Pulse out 2<br>Timer pulse per period 2. It is phase aligned with 1588 timer clock (chip external output pin). | 0 |

**Table 14-1. eTSEC*n* Network Interface Signal Properties (continued)**

| Signal Name | Function | Reset State |
|---|---|---|
| TSEC_1588_ TRIG_OUT0 | 1588—Timer alarm 0<br>Timer current time is equal to or greater than alarm time comparator register. User reprograms the TSEC_1588_ALARM*n*_H/L register to deactivate this output (chip external output pin). | 0 |
| TSEC_1588_ TRIG_OUT1 | 1588—Timer alarm 1<br>Timer current time is equal to or greater than alarm time comparator register. User reprograms the TSEC_1588_ALARM*n*_H/L register to deactivate this output (chip external output pin). | 0 |
| SD2_TX[0:1]<br>$\overline{\text{SD2\_TX[0:1]}}$ | SGMII transmit data (and complement) | — |
| $\overline{\text{SD2\_RX[0:1]}}$<br>$\overline{\text{SD2\_RX[0:1]}}$ | SGMII receive data (and complement) | — |
| SD2_REF_CLK<br>$\overline{\text{SD2\_REF\_CLK}}$ | SGMII SerDes2 PLL reference clock (and complement) | — |

## 14.4.1 Detailed Signal Descriptions

Below is a description of the eTSEC interface signals. For RGMII mode details please refer to the Hewlett-Packard reduced gigabit media-independent interface (RGMII) specification version 1.2a, dated 9/22/2000. RMII mode details follow the RMII Consortium Specification, dated 3/20/1998. All other modes follow the IEEE 802.3 standard, 2000 Edition. Input signals not used are internally disabled. Except for TSEC*n*_GTX_CLK, output signals not used are driven low.

### Table 14-2. eTSEC Signals—Detailed Signal Descriptions

| Signal | I/O | Description | |
|---|---|---|---|
| TSEC*n*_COL | I | Collision input. The behavior of this signal is not specified while in full-duplex mode. | |
| | | **State Meaning** | Asserted/Negated—In MII mode, this signal is asserted upon detection of a collision, and must remain asserted while the collision persists. In FIFO mode this signal is used to effect flow control on the transmitter.<br>This signal is not used in the following modes:<br>• RMII<br>• GMII<br>• TBI<br>• RTBI<br>• RGMII |
| | | **Timing** | Asserted/Negated—This signal is not required to transition synchronously with TSEC*n*_TX_CLK or TSEC*n*_RX_CLK. |
| TSEC*n*_CRS | I | Carrier sense input. In TBI and RTBI modes, this signal is used as SDET (signal detect).In TBI mode SDET must be tied high externally on the board. In RTBI mode SDET is tied high internally.<br>This signal is not used in the following modes:<br>• RMII<br>• GMII<br>• RGMII | |
| | | **State Meaning** | Asserted/Negated—In MII mode, TSEC*n*_CRS is asserted while the transmit or receive medium is not idle. In the event of a collision, TSEC*n*_CRS must remain asserted for the duration of the collision. |
| | | **Timing** | Asserted/Negated—This signal is not required to transition synchronously with TSEC*n*_TX_CLK or TSEC*n*_RX_CLK. |
| | O | Receiver flow control signal in FIFO mode.<br>This signal is not used in the eTSEC Ethernet modes. | |
| | | **State Meaning** | Asserted/Negated—TSEC*n*_CRS is asserted while the FIFO receiver is unprepared to accept additional receive data. |
| | | **Timing** | Asserted/Negated—This signal transitions synchronously with TSEC*n*_RX_CLK. |
| TSEC*n*_GTX_CLK | O | Gigabit transmit clock. This signal is an output from the eTSEC into the PHY. TSEC*n*_GTX_CLK is a 125-MHz clock that provides a timing reference for TX_EN, TXD, and TX_ER in the following modes:<br>• GMII<br>• TBI<br>• RTBI<br>In RGMII mode, TSEC*n*_GTX_CLK becomes the transmit clock and provides timing reference during 1000Base-T (125 MHz), 100Base-T (25 MHz) and 10Base-T (2.5 MHz) transmissions.<br>This signal feeds back the uninverted transmit clock in MII or FIFO modes, but feeds back an inverted transmit clock in RTBI or RGMII modes.<br>This signal is driven low unless transmission is enabled, or the eTSEC is in TBI or FIFO mode. | |

### Table 14-2. eTSEC Signals—Detailed Signal Descriptions (continued)

| Signal | I/O | Description |
|---|---|---|
| EC_GTX_CLK125 | I | Gigabit transmit 125-MHz source. This signal must be generated externally with a crystal or oscillator, or is sometimes provided by the PHY. EC_GTX_CLK125 is a 125-MHz input into the eTSEC and is used to generate all 125-MHz related signals and clocks in the following modes:<br>• GMII<br>• TBI<br>• RTBI<br>• RGMII<br>This input is not used in these modes:<br>• FIFO<br>• RMII<br>• SGMII<br>• MII |
| EC_MDC | O | Management data clock.<br>This signal is a clock (typically 2.5 MHz) supplied by the MAC<br>(IEEE set minimum period of 400 ns or a frequency of 2.5 MHz, but the device may be configured up to 12.5 MHz if supported by the PHY at that speed.) The frequency can be modified by writing to MIIMCFG[28:31] of the eTSEC1 controller. |
| EC_MDIO | I/O | Management data input/output. |
| | | **State Meaning** Asserted/Negated—EC_MDIO is a bidirectional signal to input PHY-supplied status during management read cycles and output control during MII management write cycles. Addressed using eTSEC1 memory-mapped registers. |
| | | **Timing** Asserted/Negated—This signal is required to be synchronous with the EC_MDC signal. |
| TSEC*n*_RX_CLK | I | Receive clock. In GMII, MII, or RGMII mode, the receive clock TSEC*n*_RX_CLK is a continuous clock (2.5, 25, or 125 MHz) that provides a timing reference for TSEC*n*_RX_DV, TSEC*n*_RXD, and TSEC*n*_RX_ER.<br>In TBI mode, TSEC*n*_RX_CLK is the input for a 62.5 MHz PMA receive clock, 0 split phase with PMA_RX_CLK1 and is supplied by the SerDes.<br>In RTBI mode it is a 125-MHz receive clock.<br>In RMII mode this clock is not used for the receive clock, as RMII uses a shared reference clock.<br>In FIFO mode the receive clock is a continuous clock. See the device hardware specifications document for maximum supported frequencies. |
| TSEC*n*_RX_DV | I | Receive data valid. In GMII or MII mode, if TSEC*n*_RX_DV is asserted, the PHY is indicating that valid data is present on the GMII and MII interfaces.<br>In RGMII mode, TSEC*n*_RX_DV becomes RX_CTL. The RX_DV and RX_ERR are received on this signal on the rising and falling edges of TSEC*n*_RX_CLK.<br>In TBI mode, TSEC*n*_RX_DV represents receive code group (RCG) bit 8. Together, with RCG[9] and RCG[7:0], they represents the 10-bit encoded symbol of GMII receive signals.<br>In RTBI mode, TSEC*n*_RX_DV represents receive code group (RCG) bit 4 and 9. On the positive edge of the TSEC*n*_RX_CLK, RCG[4] and RCG[3:0] represent the first half of the 10-bit encoded symbol. On the negative edge of the TSEC*n*_RX_CLK, RCG[9] and RCG[8:5] represent the second half of the 10-bit encoded symbol.<br>In RMII mode the PHY asserts TSEC*n*_RX_DV (CRS_DV) when the receive medium is non-idle. This signal asserts asynchronously with respect to the RMII reference clock, but negates synchronously to indicate loss of carrier.In FIFO mode TSEC*n*_RX_DV is used to indicate valid data (GMII-style protocols) or forms part of the receive control flags (encoded packet protocols). |

**Table 14-2. eTSEC Signals—Detailed Signal Descriptions (continued)**

| Signal | I/O | Description |
|---|---|---|
| TSEC*n*_RXD[7:0] | I | Receive data in. In GMII mode, TSEC*n*_RXD[7:4] with TSEC*n*_RXD[3:0], represent one complete octet of data to be transferred from the PHY to the MAC when TSEC*n*_RX_DV is asserted.<br>In TBI mode, TSEC*n*_RXD[7:4] represents RCG[7:4]. Together, with RCG[9:8] and RCG[3:0], they represent the 10-bit encoded symbol of GMII receive signals.<br>In GMII or MII mode, TSEC*n*_RXD[3:0] represents a nibble of data to be transferred from the PHY to the MAC when TSEC*n*_RX_DV is asserted. A completely-formed SFD must be passed across the MII. While TSEC*n*_RX_DV is not asserted, TSEC*n*_RXD has no meaning.<br>In RGMII mode, data bits 3:0 are received on the rising edge of TSECn_RX_CLK.<br>In RTBI mode, TSEC*n*_RXD[3:0] represents RCG[3:0] on the rising edge of TSEC*n*_RX_CLK and RCG[8:5] are received on the falling edge of TSEC*n*_RX_CLK.<br>In TBI mode, TSEC*n*_RXD[3:0] represents RCG[3:0]. Together, with RCG[9:4], they represent the 10-bit encoded symbol of GMII receive signals.<br>In RMII mode TSEC*n*_RXD[1:0] represents RXD[1:0], which is considered valid when TSEC*n*_RX_DV (CRS_DV) is asserted, or invalid otherwise.In FIFO mode TSEC*n*_RXD[7:4] with TSEC*n*_RXD[3:0] represent one complete octet of data to be received from the external FIFO device. |
| TSEC*n*_RX_ER | I | Receive error<br>**State Meaning** Asserted/Negated—In GMII, MII, or RMII mode, if TSEC*n*_RX_ER and TSEC*n*_RX_DV are asserted, the PHY has detected an error in the current frame.<br>In TBI mode, this signal represents RCG[9]. Together, with RCG[8:0], they represent the 10-bit encoded symbol of GMII receive signals.<br>In FIFO mode, this signal represents either receive data error (GMII-style protocols) or forms part of the receive control flags (encoded packet protocols). This signal is not used in the RTBI or RGMII modes. |
| TSEC*n*_TX_CLK | I | Transmit clock in. In MII mode, TSEC*n*_TX_CLK is a continuous clock (2.5 or 25 MHz) that provides a timing reference for the TSEC*n*_TX_EN, TSEC*n*_TXD, and TSEC*n*_TX_ER signals.<br>In GMII mode, this signal provides the 2.5 or 25 MHz timing reference during 10Base-T and 100Base-T and comes from the PHY. In 1000Base-T this clock is not used and TSEC*n*_GTX_CLK (125 MHz) becomes the timing reference. The TSEC*n*_GTX_CLK is generated in the eTSEC and provided to the PHY and the MAC. The TSEC*n*_TX_CLK is generated in the PHY and provided to the MAC.<br>In TBI mode, this signal is PMA receive clock 1 at 62.5 MHz, split phase with PMA_RX_CLK0, and is supplied by the SerDes.<br>In RMII mode this signal is the reference clock shared between transmit and receive, and is supplied by the PHY.<br>In FIFO mode the transmit clock is a continuous clock. See the device hardware specifications document for maximum supported frequencies.<br>This signal is not used in the eTSEC RTBI or RGMII modes. |

Table 14-2. eTSEC Signals—Detailed Signal Descriptions (continued)

| Signal | I/O | Description |
|---|---|---|
| TSEC*n*_TXD[7:0] | O | Transmit data out. In GMII mode, TSEC*n*_TXD[7:0] represents one complete octet of data to be sent from the MAC to the PHY when TSEC_TX_DV is asserted and has no meaning while TSEC*n*_TX_EN is negated.<br>In TBI mode, TSEC*n*_TXD[7:4] represents transmit code group (TCG) bits 7:4. Together, with TCG[9:8] and TCG[3:0], they represent the 10-bit encoded symbol.<br>In GMII or MII mode, TSEC*n*_TXD[3:0] represent a nibble of data to be sent from the MAC to the PHY when TSEC*n*_TX_EN is asserted and have no meaning while TSEC*n*_TX_EN is negated.<br>In RGMII or RTBI mode, data bits 3:0 are transmitted on the rising edge of TSEC*n*_TX_CLK, and data bits 7:4 are transmitted on the falling edge of TSEC*n*_TX_CLK.<br>In TBI mode, TSEC*n*_TXD[3:0] represents TCG[3:0]. Together, with TCG[9:4], they represent the 10-bit encoded symbol.<br>In RMII mode, TSEC*n*_TXD[1:0] represents TXD[1:0], which is valid data sent to the PHY when TSEC*n*_TX_EN is asserted, or undefined otherwise.<br>In FIFO mode, TSEC*n*_TXD[7:4] with TSEC*n*_TXD[3:0] represent one complete octet of data to be received from the external FIFO device.<br>Note that some of these signals are also used during reset to configure the eTSEC interface mode. |
| TSEC*n*_TX_EN | O | Transmit data valid. In GMII, MII, or RMII mode, if TSEC*n*_TX_EN is asserted, the MAC is indicating that valid data is present on the GMII's or the MII's TSEC*n*_TXD signals.<br>In RGMII mode, TSEC*n*_TX_EN becomes TX_CTL. TX_EN and TX_ERR are asserted on this signal on rising and falling edges of the TSEC*n*_GTX_CLK, respectively.<br>In TBI mode, TSEC*n*_TX_EN represents TCG[8]. Together, with TCG[9] and TCG[7:0], they represent the 10-bit encoded symbol.<br>In RTBI mode, TSEC*n*_TX_EN represents TCG[4] on the rising edge and TCG[9] on the falling edge of TSEC*n*_GTX_CLK, respectively. Together with TCG[3:0] and TCG[8:5], they represent the 10-bit encoded symbol.<br>In FIFO mode TSEC*n*_TX_EN is used to indicate valid data (GMII-style protocols) or forms part of the transmit control flags (encoded packet protocols). |
| TSEC*n*_TX_ER | O | Transmit error. In GMII or MII mode, assertion of TSEC*n*_TX_ER for one or more clock cycles while TSEC*n*_TX_EN is asserted causes the PHY to transmit one or more illegal symbols. Asserting TSEC*n*_TX_ER has no effect while operating at 10 Mbps or while TSEC*n*_TX_EN is negated. This signal transitions synchronously with respect to TSEC*n*_TX_CLK.<br>In TBI mode, TSEC*n*_TX_ER represents TCG[9]. Together, with TCG[8:0], they represents the 10-bit encoded symbol.<br>In FIFO mode TSEC*n*_TX_ER represents either transmit data error (GMII-style protocols) or forms part of the transmit control flags (encoded packet protocols).<br>This signal is not used in the eTSEC RMII, RTBI, or RGMII modes and is driven low. |
| TSEC_1588_CLK | I | 1588 clock in. External high precision timer reference clock input (chip external input pin). |
| TSEC_1588_ CLK_OUT | O | 1588 clock out. Phase aligned timer clock divider output (chip external output pin). |
| TSEC_1588_ TRIG_IN0 | I | 1588 trigger in 0. External timer trigger input 0.This is an asynchronous general purpose input (chip external input pin). |
| TSEC_1588_ TRIG_IN1 | i | 1588 trigger in 1. External timer trigger input 1.This is an asynchronous general purpose input (chip external input pin). |
| TSEC_1588_ PULSE_OUT1 | O | 1588 pulse out 1. Timer pulse per period 1. It is phase aligned with 1588 timer clock (chip external output pin) |
| TSEC_1588_ PULSE_OUT2 | O | 1588 pulse out 2. Timer pulse per period 2. It is phase aligned with 1588 timer clock (chip external output pin) |

**Table 14-2. eTSEC Signals—Detailed Signal Descriptions (continued)**

| Signal | I/O | Description |
|---|---|---|
| TSEC_1588_ TRIG_OUT0 | O | 1588 timer alarm 0. Timer current time is equal to or greater than alarm time comparator register. User reprograms the TSEC_1588_ALARM*n*_H/L register to deactivate this output (chip external output pin) |
| TSEC_1588_ TRIG_OUT1 | O | 1588 timer alarm 1. Timer current time is equal to or greater than alarm time comparator register. User reprograms the TSEC_1588_ALARM*n*_H/L register to deactivate this output (chip external output pin) |
| SD2_TX[0:1]<br>$\overline{\text{SD2\_TX[0:1]}}$ | O | SGMII transmit data (and complement)<br>When in SGMII interface mode:<br>• eTSEC1 utilizes SD2_TX[0] and $\overline{\text{SD2\_TX[0]}}$<br>• eTSEC3 utilizes SD2_TX[2] and $\overline{\text{SD2\_TX[2]}}$ |
| SD2_RX[0:1]<br>$\overline{\text{SD2\_RX[0:1]}}$ | I | SGMII receive data (and complement)<br>When in SGMII interface mode:<br>• eTSEC1 utilizes SD2_RX[0] and $\overline{\text{SD2\_RX[0]}}$<br>• eTSEC3 utilizes SD2_RX[2] and $\overline{\text{SD2\_RX[2]}}$ |
| SD2_REF_CLK<br>$\overline{\text{SD2\_REF\_CLK}}$ | I | SGMII SerDes2 PLL reference clock (and complement) |

## 14.5 Memory Map/Register Definition

The eTSECs use a software model that is a superset of the PowerQUICC III TSEC functionality and is similar to that employed by the Fast Ethernet function supported on the Freescale MPC8260 CPM FCC and in the FEC of the MPC860T.

The eTSEC device is programmed by a combination of control/status registers (CSRs) and buffer descriptors. The CSRs are used for mode control, interrupts, and to extract status information. The descriptors are used to pass data buffers and related buffer status or frame information between the hardware and software.

All accesses to and from the registers must be made as 32-bit accesses. There is no support for accesses of sizes other than 32 bits. Writes to reserved register bits must always store 0, as writing 1 to reserved bits may have unintended side-effects. Reads from unmapped register addresses return zero. Unless otherwise specified, the read value of reserved bits in mapped registers is not defined, and must not be assumed to be 0.

This section of the document defines the memory map and describes the registers in detail. The buffer descriptor is described in Section 14.6.8, "Buffer Descriptors."

The ten-bit interface (TBI) and reduced ten-bit interface (RTBI) module MII registers are also described in this section. The TBI/RTBI registers are defined like PHY registers and, as such, are accessed through the MII management interface in the same way the PHYs are accessed. For detailed descriptions of the TBI/RTBI registers (the MII register set for the ten-bit interface) refer to Section 14.5.4, "Ten-Bit Interface (TBI)."

### 14.5.1 Top-Level Module Memory Map

Each of the eTSECs is allocated 4 Kbytes of memory-mapped space. The space for each eTSEC is divided as indicated in Table 14-3.

Table 14-3. Module Memory Map Summary

| Address Offset | Function |
|---|---|
| 000–0FF | eTSEC general control/status registers |
| 100–2FF | eTSEC transmit control/status registers |
| 300–4FF | eTSEC receive control/status registers |
| 500–5FF | MAC registers |
| 600–7FF | RMON MIB registers |
| 800–8FF | Hash table registers |
| 900–9FF | — |
| A00–AFF | FIFO control/status registers |
| B00–BFF | DMA system registers |
| C00–C3F | Lossless Flow Control registers |
| C40–DFF | — |
| E00–EFF | 1588 Hardware Assist |

### 14.5.2 Detailed Memory Map

The eTSEC memory mapped registers are accessed by reading and writing to an address comprised of the base address (specified in CCSRBAR as defined in Chapter 2, “Memory Map.”) plus the block base address, plus the offset of the specific register to be accessed. Note that all memory-mapped registers must only be accessed as 32-bit quantities.

Table 14-4 lists the offset, name, and a cross-reference to the complete description of each register. The offsets to the memory map table are applicable to each eTSEC. Block base addresses are as follows:

- eTSEC1 starts at 0x2_4000 address offset
- eTSEC3 starts at 0x2_6000 address offset

In this table and in the register figures and field descriptions, the following access definitions apply:

- Reserved fields are always ignored for the purposes of determining access type.
- R/W, R, and W (read/write, read only, and write only) indicate that all the non-reserved fields in a register have the same access type.
- w1c indicates that all of the non-reserved fields in a register are cleared by writing ones to them.
- Mixed indicates a combination of access types.
- Special is used when no other category applies. In this case the register figure and field description table should be read carefully.

### Table 14-4. Module Memory Map

| eTSEC1 Offset | Name[1] | Access [2] | Reset | Section/Page |
|---|---|---|---|---|
| | **eTSEC General Control and Status Registers** | | | |
| 0x2_4000 | TSEC_ID*—Controller ID register | R | ~~0x0124_0000~~ | 14.5.3.1.1/14-26 |
| 0x2_4004 | TSEC_ID2*—Controller ID register | R | ~~0x0030_00F0~~ | 14.5.3.1.2/14-27 |
| 0x2_4008–<br>0x2_400C | Reserved | — | — | — |
| 0x2_4010 | IEVENT—Interrupt event register | w1c | 0x0000_0000 | 14.5.3.1.3/14-27 |
| 0x2_4014 | IMASK—Interrupt mask register | R/W | 0x0000_0000 | 14.5.3.1.4/14-31 |
| 0x2_4018 | EDIS—Error disabled register | R/W | 0x0000_0000 | 14.5.3.1.5/14-33 |
| 0x2_401C | Reserved | — | — | — |
| 0x2_4020 | ECNTRL—Ethernet control register | R/W | 0x0000_0000 | 14.5.3.1.6/14-35 |
| 0x2_4024 | Reserved | — | — | — |
| 0x2_4028 | PTV—Pause time value register | R/W | 0x0000_0000 | 14.5.3.1.7/14-37 |
| 0x2_402C | DMACTRL—DMA control register | R/W | 0x0000_0000 | 14.5.3.1.8/14-38 |
| 0x2_4030 | TBIPA—TBI PHY address register | R/W | 0x0000_0000 | 14.5.3.1.9/14-40 |
| 0x2_4034–<br>0x2_40FC | Reserved | — | — | — |
| | **eTSEC Transmit Control and Status Registers** | | | |
| 0x2_4100 | TCTRL—Transmit control register | R/W | 0x0000_0000 | 14.5.3.2.1/14-40 |
| 0x2_4104 | TSTAT—Transmit status register | w1c | 0x0000_0000 | 14.5.3.2.2/14-42 |
| 0x2_4108 | DFVLAN*—Default VLAN control word | R/W | 0x8100_0000 | 14.5.3.2.3/14-46 |
| 0x2_410C | Reserved | — | — | — |
| 0x2_4110 | TXIC—Transmit interrupt coalescing register | R/W | 0x0000_0000 | 14.5.3.2.4/14-47 |
| 0x2_4114 | TQUEUE*—Transmit queue control register | R/W | 0x0000_8000 | 14.5.3.2.5/14-48 |
| 0x2_4118–<br>0x2_413C | Reserved | — | — | — |
| 0x2_4140 | TR03WT*—TxBD Rings 0–3 round-robin weightings | R/W | 0x0000_0000 | 14.5.3.2.6/14-49 |
| 0x2_4144 | TR47WT*—TxBD Rings 4–7 round-robin weightings | R/W | 0x0000_0000 | 14.5.3.2.7/14-49 |
| 0x2_4148–<br>0x2_417C | Reserved | — | — | — |
| 0x2_4180 | TBDBPH*—Tx data buffer pointer high bits | R/W | 0x0000_0000 | 14.5.3.2.8/14-50 |
| 0x2_4184 | TBPTR0—TxBD pointer for ring 0 | R/W | 0x0000_0000 | 14.5.3.2.9/14-50 |
| 0x2_4188 | Reserved | — | — | — |
| 0x2_418C | TBPTR1*—TxBD pointer for ring 1 | R/W | 0x0000_0000 | 14.5.3.2.9/14-50 |
| 0x2_4190 | Reserved | — | — | — |
| 0x2_4194 | TBPTR2*—TxBD pointer for ring 2 | R/W | 0x0000_0000 | 14.5.3.2.9/14-50 |

Table 14-4. Module Memory Map (continued)

| eTSEC1 Offset | Name[1] | Access [2] | Reset | Section/Page |
|---|---|---|---|---|
| 0x2_4198 | Reserved | — | — | — |
| 0x2_419C | TBPTR3*—TxBD pointer for ring 3 | R/W | 0x0000_0000 | 14.5.3.2.9/14-50 |
| 0x2_41A0 | Reserved | — | — | — |
| 0x2_41A4 | TBPTR4*—TxBD pointer for ring 4 | R/W | 0x0000_0000 | 14.5.3.2.9/14-50 |
| 0x2_41A8 | Reserved | — | — | — |
| 0x2_41AC | TBPTR5*—TxBD pointer for ring 5 | R/W | 0x0000_0000 | 14.5.3.2.9/14-50 |
| 0x2_41B0 | Reserved | — | — | — |
| 0x2_41B4 | TBPTR6*—TxBD pointer for ring 6 | R/W | 0x0000_0000 | 14.5.3.2.9/14-50 |
| 0x2_41B8 | Reserved | — | — | — |
| 0x2_41BC | TBPTR7*—TxBD pointer for ring 7 | R/W | 0x0000_0000 | 14.5.3.2.9/14-50 |
| 0x2_41C0–<br>0x2_41FC | Reserved | — | — | — |
| 0x2_4200 | TBASEH*—TxBD base address high bits | R/W | 0x0000_0000 | 14.5.3.2.10/14-51 |
| 0x2_4204 | TBASE0—TxBD base address of ring 0 | R/W | 0x0000_0000 | 14.5.3.2.11/14-52 |
| 0x2_4208 | Reserved | — | — | — |
| 0x2_420C | TBASE1*—TxBD base address of ring 1 | R/W | 0x0000_0000 | 14.5.3.2.11/14-52 |
| 0x2_4210 | Reserved | — | — | — |
| 0x2_4214 | TBASE2*—TxBD base address of ring 2 | R/W | 0x0000_0000 | 14.5.3.2.11/14-52 |
| 0x2_4218 | Reserved | — | — | — |
| 0x2_421C | TBASE3*—TxBD base address of ring 3 | R/W | 0x0000_0000 | 14.5.3.2.11/14-52 |
| 0x2_4220 | Reserved | — | — | — |
| 0x2_4224 | TBASE4*—TxBD base address of ring 4 | R/W | 0x0000_0000 | 14.5.3.2.11/14-52 |
| 0x2_4228 | Reserved | — | — | — |
| 0x2_422C | TBASE5*—TxBD base address of ring 5 | R/W | 0x0000_0000 | 14.5.3.2.11/14-52 |
| 0x2_4230 | Reserved | — | — | — |
| 0x2_4234 | TBASE6*—TxBD base address of ring 6 | R/W | 0x0000_0000 | 14.5.3.2.11/14-52 |
| 0x2_4238 | Reserved | — | — | — |
| 0x2_423C | TBASE7*—TxBD base address of ring 7 | R/W | 0x0000_0000 | 14.5.3.2.11/14-52 |
| 0x2_4240–<br>0x2_427C | Reserved | — | — | — |
| 0x2_4280 | TMR_TXTS1_ID* - Tx time stamp identification tag (set 1) | R/W | 0x0000_0000 | 14.5.3.2.12/14-52 |
| 0x2_4284 | TMR_TXTS2_ID* - Tx time stamp identification tag (set 2) | R/W | 0x0000_0000 | 14.5.3.2.12/14-52 |
| 0x2_4288–<br>0x2_42BC | Reserved | — | — | — |
| 0x2_42C0 | TMR_TXTS1_H* - Tx time stamp high (set 1) | R/W | 0x0000_0000 | 14.5.3.2.13/14-53 |

Table 14-4. Module Memory Map (continued)

| eTSEC1 Offset | Name[1] | Access [2] | Reset | Section/Page |
|---|---|---|---|---|
| 0x2_42C4 | TMR_TXTS1_L* - Tx time stamp high (set 1) | R/W | 0x0000_0000 | 14.5.3.2.13/14-53 |
| 0x2_42C8 | TMR_TXTS2_H* - Tx time stamp high (set 2) | R/W | 0x0000_0000 | 14.5.3.2.13/14-53 |
| 0x2_42CC | TMR_TXTS2_L* - Tx time stamp high (set 2) | R/W | 0x0000_0000 | 14.5.3.2.13/14-53 |
| 0x2_42D0–0x2_42FC | Reserved | — | — | — |
| **eTSEC Receive Control and Status Registers** | | | | |
| 0x2_4300 | RCTRL—Receive control register | R/W | 0x0000_0000 | 14.5.3.3.1/14-54 |
| 0x2_4304 | RSTAT—Receive status register | w1c | 0x0000_0000 | 14.5.3.3.2/14-56 |
| 0x2_4308–0x2_430C | Reserved | — | — | — |
| 0x2_4310 | RXIC—Receive interrupt coalescing register | R/W | 0x0000_0000 | 14.5.3.3.3/14-59 |
| 0x2_4314 | RQUEUE*—Receive queue control register. | R/W | 0x0080_0080 | 14.5.3.3.4/14-60 |
| 0x2_4318–0x2_432C | Reserved | — | — | — |
| 0x2_4330 | RBIFX*—Receive bit field extract control register | R/W | 0x0000_0000 | 14.5.3.3.5/14-61 |
| 0x2_4334 | RQFAR*—Receive queue filing table address register | R/W | 0x0000_0000 | 14.5.3.3.6/14-63 |
| 0x2_4338 | RQFCR*—Receive queue filing table control register | R/W | 0x*nnnn_nnnn* | 14.5.3.3.7/14-63 |
| 0x2_433C | RQFPR*—Receive queue filing table property register | R/W | 0x*nnnn_nnnn* | 14.5.3.3.8/14-65 |
| 0x2_4340 | MRBLR—Maximum receive buffer length register | R/W | 0x0000_0000 | 14.5.3.3.9/14-68 |
| 0x2_4344–0x2_437C | Reserved | — | — | — |
| 0x2_4380 | RBDBPH*—Rx data buffer pointer high bits | R/W | 0x0000_0000 | 14.5.3.3.10/14-68 |
| 0x2_4384 | RBPTR0—RxBD pointer for ring 0 | R/W | 0x0000_0000 | 14.5.3.3.11/14-69 |
| 0x2_4388 | Reserved | — | — | — |
| 0x2_438C | RBPTR1*—RxBD pointer for ring 1 | R/W | 0x0000_0000 | 14.5.3.3.11/14-69 |
| 0x2_4390 | Reserved | — | — | — |
| 0x2_4394 | RBPTR2*—RxBD pointer for ring 2 | R/W | 0x0000_0000 | 14.5.3.3.11/14-69 |
| 0x2_4398 | Reserved | — | — | — |
| 0x2_439C | RBPTR3*—RxBD pointer for ring 3 | R/W | 0x0000_0000 | 14.5.3.3.11/14-69 |
| 0x2_43A0 | Reserved | — | — | — |
| 0x2_43A4 | RBPTR4*—RxBD pointer for ring 4 | R/W | 0x0000_0000 | 14.5.3.3.11/14-69 |
| 0x2_43A8 | Reserved | — | — | — |
| 0x2_43AC | RBPTR5*—RxBD pointer for ring 5 | R/W | 0x0000_0000 | 14.5.3.3.11/14-69 |
| 0x2_43B0 | Reserved | — | — | — |
| 0x2_43B4 | RBPTR6*—RxBD pointer for ring 6 | R/W | 0x0000_0000 | 14.5.3.3.11/14-69 |

Table 14-4. Module Memory Map (continued)

| eTSEC1 Offset | Name[1] | Access [2] | Reset | Section/Page |
|---|---|---|---|---|
| 0x2_43B8 | Reserved | — | — | — |
| 0x2_43BC | RBPTR7*—RxBD pointer for ring 7 | R/W | 0x0000_0000 | 14.5.3.3.11/14-69 |
| 0x2_43C0–<br>0x2_43FC | Reserved | — | — | — |
| 0x2_4400 | RBASEH*—RxBD base address high bits | R/W | 0x0000_0000 | 14.5.3.3.12/14-70 |
| 0x2_4404 | RBASE0—RxBD base address of ring 0 | R/W | 0x0000_0000 | 14.5.3.3.13/14-70 |
| 0x2_4408 | Reserved | — | — | — |
| 0x2_440C | RBASE1*—RxBD base address of ring 1 | R/W | 0x0000_0000 | 14.5.3.3.13/14-70 |
| 0x2_4410 | Reserved | — | — | — |
| 0x2_4414 | RBASE2*—RxBD base address of ring 2 | R/W | 0x0000_0000 | 14.5.3.3.13/14-70 |
| 0x2_4418 | Reserved | — | — | — |
| 0x2_441C | RBASE3*—RxBD base address of ring 3 | R/W | 0x0000_0000 | 14.5.3.3.13/14-70 |
| 0x2_4420 | Reserved | — | — | — |
| 0x2_4424 | RBASE4*—RxBD base address of ring 4 | R/W | 0x0000_0000 | 14.5.3.3.13/14-70 |
| 0x2_4428 | Reserved | — | — | — |
| 0x2_442C | RBASE5*—RxBD base address of ring 5 | R/W | 0x0000_0000 | 14.5.3.3.13/14-70 |
| 0x2_4430 | Reserved | — | — | — |
| 0x2_4434 | RBASE6*—RxBD base address of ring 6 | R/W | 0x0000_0000 | 14.5.3.3.13/14-70 |
| 0x2_4438 | Reserved | — | — | — |
| 0x2_443C | RBASE7*—RxBD base address of ring 7 | R/W | 0x0000_0000 | 14.5.3.3.13/14-70 |
| 0x2_4440–<br>0x2_44BC | Reserved | — | — | — |
| 0x2_44C0 | TMR_RXTS_H* - Rx timer time stamp register high | R/W | 0x0000_0000 | 14.5.3.3.14/14-71 |
| 0x2_44C4 | TMR_RXTS_L* - Rx timer time stamp register low | R/W | 0x0000_0000 | 14.5.3.3.14/14-71 |
| 0x2_44C8–<br>0x2_44FC | Reserved | — | — | — |
| eTSEC MAC Registers | | | | |
| 0x2_4500 | MACCFG1—MAC configuration register 1 | R/W | 0x0000_0000 | 14.5.3.5.1/14-74 |
| 0x2_4504 | MACCFG2—MAC configuration register 2 | R/W | 0x0000_7000 | 14.5.3.5.2/14-76 |
| 0x2_4508 | IPGIFG—Inter-packet/inter-frame gap register | R/W | 0x4060_5060 | 14.5.3.5.3/14-78 |
| 0x2_450C | HAFDUP—Half-duplex control | R/W | 0x00A1_F037 | 14.5.3.5.4/14-79 |
| 0x2_4510 | MAXFRM—Maximum frame length | R/W | 0x0000_0600 | 14.5.3.5.5/14-80 |
| 0x2_4514–<br>0x2_451C | Reserved | — | — | — |
| 0x2_4520 | MIIMCFG—MII management configuration | R/W | 0x0000_0007 | 14.5.3.5.6/14-80 |

Table 14-4. Module Memory Map (continued)

| eTSEC1 Offset | Name[1] | Access [2] | Reset | Section/Page |
|---|---|---|---|---|
| 0x2_4524 | MIIMCOM—MII management command | R/W | 0x0000_0000 | 14.5.3.5.7/14-81 |
| 0x2_4528 | MIIMADD—MII management address | R/W | 0x0000_0000 | 14.5.3.5.8/14-82 |
| 0x2_452C | MIIMCON—MII management control | WO | 0x0000_0000 | 14.5.3.5.9/14-82 |
| 0x2_4530 | MIIMSTAT—MII management status | R | 0x0000_0000 | 14.5.3.5.10/14-83 |
| 0x2_4534 | MIIMIND—MII management indicator | R | 0x0000_0000 | 14.5.3.5.11/14-83 |
| 0x2_4538 | Reserved | — | — | — |
| 0x2_453C | IFSTAT—Interface status | R | 0x0000_0000 | 14.5.3.5.12/14-84 |
| 0x2_4540 | MACSTNADDR1—MAC station address register 1 | R/W | 0x0000_0000 | 14.5.3.5.13/14-84 |
| 0x2_4544 | MACSTNADDR2—MAC station address register 2 | R/W | 0x0000_0000 | 14.5.3.5.14/14-85 |
| 0x2_4548 | MAC01ADDR1*—MAC exact match address 1, part 1 | R/W | 0x0000_0000 | 14.5.3.5.15/14-86<br>14.5.3.5.16/14-86 |
| 0x2_454C | MAC01ADDR2*—MAC exact match address 1, part 2 | R/W | 0x0000_0000 | |
| 0x2_4550 | MAC02ADDR1*—MAC exact match address 2, part 1 | R/W | 0x0000_0000 | |
| 0x2_4554 | MAC02ADDR2*—MAC exact match address 2, part 2 | R/W | 0x0000_0000 | |
| 0x2_4558 | MAC03ADDR1*—MAC exact match address 3, part 1 | R/W | 0x0000_0000 | |
| 0x2_455C | MAC03ADDR2*—MAC exact match address 3, part 2 | R/W | 0x0000_0000 | |
| 0x2_4560 | MAC04ADDR1*—MAC exact match address 4, part 1 | R/W | 0x0000_0000 | |
| 0x2_4564 | MAC04ADDR2*—MAC exact match address 4, part 2 | R/W | 0x0000_0000 | |
| 0x2_4568 | MAC05ADDR1*—MAC exact match address 5, part 1 | R/W | 0x0000_0000 | |
| 0x2_456C | MAC05ADDR2*—MAC exact match address 5, part 2 | R/W | 0x0000_0000 | |

Table 14-4. Module Memory Map (continued)

| eTSEC1 Offset | Name[1] | Access [2] | Reset | Section/Page |
|---|---|---|---|---|
| 0x2_4570 | MAC06ADDR1*—MAC exact match address 6, part 1 | R/W | 0x0000_0000 | 14.5.3.5.15/14-86<br>14.5.3.5.16/14-86 |
| 0x2_4574 | MAC06ADDR2*—MAC exact match address 6, part 2 | R/W | 0x0000_0000 | |
| 0x2_4578 | MAC07ADDR1*—MAC exact match address 7, part 1 | R/W | 0x0000_0000 | |
| 0x2_457C | MAC07ADDR2*—MAC exact match address 7, part 2 | R/W | 0x0000_0000 | |
| 0x2_4580 | MAC08ADDR1*—MAC exact match address 8, part 1 | R/W | 0x0000_0000 | |
| 0x2_4584 | MAC08ADDR2*—MAC exact match address 8, part 2 | R/W | 0x0000_0000 | |
| 0x2_4588 | MAC09ADDR1*—MAC exact match address 9, part 1 | R/W | 0x0000_0000 | |
| 0x2_458C | MAC09ADDR2*—MAC exact match address 9, part 2 | R/W | 0x0000_0000 | |
| 0x2_4590 | MAC10ADDR1*—MAC exact match address 10, part 1 | R/W | 0x0000_0000 | |
| 0x2_4594 | MAC10ADDR2*—MAC exact match address 10, part 2 | R/W | 0x0000_0000 | |
| 0x2_4598 | MAC11ADDR1*—MAC exact match address 11, part 1 | R/W | 0x0000_0000 | |
| 0x2_459C | MAC11ADDR2*—MAC exact match address 11, part 2 | R/W | 0x0000_0000 | |
| 0x2_45A0 | MAC12ADDR1*—MAC exact match address 12, part 1 | R/W | 0x0000_0000 | |
| 0x2_45A4 | MAC12ADDR2*—MAC exact match address 12, part 2 | R/W | 0x0000_0000 | |
| 0x2_45A8 | MAC13ADDR1*—MAC exact match address 13, part 1 | R/W | 0x0000_0000 | |
| 0x2_45AC | MAC13ADDR2*—MAC exact match address 13, part 2 | R/W | 0x0000_0000 | |
| 0x2_45B0 | MAC14ADDR1*—MAC exact match address 14, part 1 | R/W | 0x0000_0000 | |
| 0x2_45B4 | MAC14ADDR2*—MAC exact match address 14, part 2 | R/W | 0x0000_0000 | |
| 0x2_45B8 | MAC15ADDR1*—MAC exact match address 15, part 1 | R/W | 0x0000_0000 | |
| 0x2_45BC | MAC15ADDR2*—MAC exact match address 15, part 2 | R/W | 0x0000_0000 | |
| 0x2_45C0–0x2_467C | Reserved | — | — | — |
| **eTSEC Transmit and Receive Counters** | | | | |
| 0x2_4680 | TR64—Transmit and receive 64-byte frame counter | R/W | 0x0000_0000 | 14.5.3.6.1/14-88 |
| 0x2_4684 | TR127—Transmit and receive 65- to 127-byte frame counter | R/W | 0x0000_0000 | 14.5.3.6.2/14-88 |
| 0x2_4688 | TR255—Transmit and receive 128- to 255-byte frame counter | R/W | 0x0000_0000 | 14.5.3.6.3/14-89 |
| 0x2_468C | TR511—Transmit and receive 256- to 511-byte frame counter | R/W | 0x0000_0000 | 14.5.3.6.4/14-89 |
| 0x2_4690 | TR1K—Transmit and receive 512- to 1023-byte frame counter | R/W | 0x0000_0000 | 14.5.3.6.5/14-90 |
| 0x2_4694 | TRMAX—Transmit and receive 1024- to 1518-byte frame counter | R/W | 0x0000_0000 | 14.5.3.6.6/14-90 |
| 0x2_4698 | TRMGV—Transmit and receive 1519- to 1522-byte good VLAN frame count | R/W | 0x0000_0000 | 14.5.3.6.7/14-91 |
| **eTSEC Receive Counters** | | | | |
| 0x2_469C | RBYT—Receive byte counter | R/W | 0x0000_0000 | 14.5.3.6.8/14-91 |
| 0x2_46A0 | RPKT—Receive packet counter | R/W | 0x0000_0000 | 14.5.3.6.9/14-92 |
| 0x2_46A4 | RFCS—Receive FCS error counter | R/W | 0x0000_0000 | 14.5.3.6.10/14-92 |
| 0x2_46A8 | RMCA—Receive multicast packet counter | R/W | 0x0000_0000 | 14.5.3.6.11/14-93 |

## Table 14-4. Module Memory Map (continued)

| eTSEC1 Offset | Name[1] | Access [2] | Reset | Section/Page |
|---|---|---|---|---|
| 0x2_46AC | RBCA—Receive broadcast packet counter | R/W | 0x0000_0000 | 14.5.3.6.12/14-93 |
| 0x2_46B0 | RXCF—Receive control frame packet counter | R/W | 0x0000_0000 | 14.5.3.6.13/14-94 |
| 0x2_46B4 | RXPF—Receive PAUSE frame packet counter | R/W | 0x0000_0000 | 14.5.3.6.14/14-94 |
| 0x2_46B8 | RXUO—Receive unknown OP code counter | R/W | 0x0000_0000 | 14.5.3.6.15/14-95 |
| 0x2_46BC | RALN—Receive alignment error counter | R/W | 0x0000_0000 | 14.5.3.6.16/14-95 |
| 0x2_46C0 | RFLR—Receive frame length error counter | R/W | 0x0000_0000 | 14.5.3.6.17/14-96 |
| 0x2_46C4 | RCDE—Receive code error counter | R/W | 0x0000_0000 | 14.5.3.6.18/14-96 |
| 0x2_46C8 | RCSE—Receive carrier sense error counter | R/W | 0x0000_0000 | 14.5.3.6.19/14-97 |
| 0x2_46CC | RUND—Receive undersize packet counter | R/W | 0x0000_0000 | 14.5.3.6.20/14-97 |
| 0x2_46D0 | ROVR—Receive oversize packet counter | R/W | 0x0000_0000 | 14.5.3.6.21/14-98 |
| 0x2_46D4 | RFRG—Receive fragments counter | R/W | 0x0000_0000 | 14.5.3.6.22/14-98 |
| 0x2_46D8 | RJBR—Receive jabber counter | R/W | 0x0000_0000 | 14.5.3.6.23/14-99 |
| 0x2_46DC | RDRP—Receive drop counter | R/W | 0x0000_0000 | 14.5.3.6.24/14-99 |
| **eTSEC Transmit Counters** | | | | |
| 0x2_46E0 | TBYT—Transmit byte counter | R/W | 0x0000_0000 | 14.5.3.6.25/14-100 |
| 0x2_46E4 | TPKT—Transmit packet counter | R/W | 0x0000_0000 | 14.5.3.6.26/14-100 |
| 0x2_46E8 | TMCA—Transmit multicast packet counter | R/W | 0x0000_0000 | 14.5.3.6.27/14-101 |
| 0x2_46EC | TBCA—Transmit broadcast packet counter | R/W | 0x0000_0000 | 14.5.3.6.28/14-101 |
| 0x2_46F0 | TXPF—Transmit PAUSE control frame counter | R/W | 0x0000_0000 | 14.5.3.6.29/14-102 |
| 0x2_46F4 | TDFR—Transmit deferral packet counter | R/W | 0x0000_0000 | 14.5.3.6.30/14-102 |
| 0x2_46F8 | TEDF—Transmit excessive deferral packet counter | R/W | 0x0000_0000 | 14.5.3.6.31/14-103 |
| 0x2_46FC | TSCL—Transmit single collision packet counter | R/W | 0x0000_0000 | 14.5.3.6.32/14-103 |
| 0x2_4700 | TMCL—Transmit multiple collision packet counter | R/W | 0x0000_0000 | 14.5.3.6.33/14-104 |
| 0x2_4704 | TLCL—Transmit late collision packet counter | R/W | 0x0000_0000 | 14.5.3.6.34/14-104 |
| 0x2_4708 | TXCL—Transmit excessive collision packet counter | R/W | 0x0000_0000 | 14.5.3.6.35/14-105 |
| 0x2_470C | TNCL—Transmit total collision counter | R/W | 0x0000_0000 | 14.5.3.6.36/14-105 |
| 0x2_4710 | Reserved | — | — | — |
| 0x2_4714 | TDRP—Transmit drop frame counter | R/W | 0x0000_0000 | 14.5.3.6.37/14-106 |
| 0x2_4718 | TJBR—Transmit jabber frame counter | R/W | 0x0000_0000 | 14.5.3.6.38/14-106 |
| 0x2_471C | TFCS—Transmit FCS error counter | R/W | 0x0000_0000 | 14.5.3.6.39/14-107 |
| 0x2_4720 | TXCF—Transmit control frame counter | R/W | 0x0000_0000 | 14.5.3.6.40/14-107 |
| 0x2_4724 | TOVR—Transmit oversize frame counter | R/W | 0x0000_0000 | 14.5.3.6.41/14-108 |
| 0x2_4728 | TUND—Transmit undersize frame counter | R/W | 0x0000_0000 | 14.5.3.6.42/14-108 |
| 0x2_472C | TFRG—Transmit fragments frame counter | R/W | 0x0000_0000 | 14.5.3.6.43/14-109 |

Table 14-4. Module Memory Map (continued)

| eTSEC1 Offset | Name[1] | Access [2] | Reset | Section/Page |
|---|---|---|---|---|
| eTSEC Counter Control and TOE Statistics Registers | | | | |
| 0x2_4730 | CAR1—Carry register one register[3] | w1c | 0x0000_0000 | 14.5.3.6.44/14-109 |
| 0x2_4734 | CAR2—Carry register two register [3] | w1c | 0x0000_0000 | 14.5.3.6.45/14-111 |
| 0x2_4738 | CAM1—Carry register one mask register | R/W | 0xFE03_FFFF | 14.5.3.6.46/14-112 |
| 0x2_473C | CAM2—Carry register two mask register | R/W | 0x000F_FFFD | 14.5.3.6.47/14-113 |
| 0x2_4740 | RREJ*—Receive filer rejected packet counter | R/W | 0x0000_0000 | 14.5.3.6.48/14-114 |
| 0x2_4744–<br>0x2_47FC | Reserved | — | — | — |
| Hash Function Registers | | | | |
| 0x2_4800 | IGADDR0—Individual/group address register 0 | R/W | 0x0000_0000 | 14.5.3.7.1/14-115 |
| 0x2_4804 | IGADDR1—Individual/group address register 1 | R/W | 0x0000_0000 | |
| 0x2_4808 | IGADDR2—Individual/group address register 2 | R/W | 0x0000_0000 | |
| 0x2_480C | IGADDR3—Individual/group address register 3 | R/W | 0x0000_0000 | |
| 0x2_4810 | IGADDR4—Individual/group address register 4 | R/W | 0x0000_0000 | |
| 0x2_4814 | IGADDR5—Individual/group address register 5 | R/W | 0x0000_0000 | |
| 0x2_4818 | IGADDR6—Individual/group address register 6 | R/W | 0x0000_0000 | |
| 0x2_481C | IGADDR7—Individual/group address register 7 | R/W | 0x0000_0000 | |
| 0x2_4820–<br>0x2_487C | Reserved | — | — | — |
| 0x2_4880 | GADDR0—Group address register 0 | R/W | 0x0000_0000 | 14.5.3.7.2/14-116 |
| 0x2_4884 | GADDR1—Group address register 1 | R/W | 0x0000_0000 | |
| 0x2_4888 | GADDR2—Group address register 2 | R/W | 0x0000_0000 | |
| 0x2_488C | GADDR3—Group address register 3 | R/W | 0x0000_0000 | |
| 0x2_4890 | GADDR4—Group address register 4 | R/W | 0x0000_0000 | |
| 0x2_4894 | GADDR5—Group address register 5 | R/W | 0x0000_0000 | |
| 0x2_4898 | GADDR6—Group address register 6 | R/W | 0x0000_0000 | |
| 0x2_489C | GADDR7—Group address register 7 | R/W | 0x0000_0000 | |
| 0x2_48A0–<br>0x2_49FC | Reserved | — | — | — |
| eTSEC FIFO Control Registers | | | | |
| 0x2_4A00 | FIFOCFG*—FIFO interface configuration register | R/W | 0x0000_00C0 | 14.5.3.8.1/14-116 |
| 0x2_4A04–<br>0x2_4AFC | Reserved | — | — | — |
| eTSEC DMA Attribute Registers | | | | |
| 0x2_4B00–<br>0x2_4BF4 | Reserved | — | — | — |

Table 14-4. Module Memory Map (continued)

| eTSEC1 Offset | Name[1] | Access [2] | Reset | Section/Page |
|---|---|---|---|---|
| 0x2_4BF8 | ATTR—Attribute register | R/W | 0x0000_0000 | 14.5.3.9.1/14-118 |
| 0x2_4BFC | ATTRELI*—Attribute extract length and extract index register | R/W | 0x0000_0000 | 14.5.3.9.2/14-119 |
| **eTSEC Lossless Flow Control Registers** | | | | |
| 0x2_4C00 | RQPRM0*—Receive Queue Parameters register 0 | R/W | 0x0000_0000 | 14.5.3.10.1/14-120 |
| 0x2_4C04 | RQPRM1*—Receive Queue Parameters register 1 | R/W | 0x0000_0000 | |
| 0x2_4C08 | RQPRM2*—Receive Queue Parameters register 2 | R/W | 0x0000_0000 | |
| 0x2_4C0C | RQPRM3*—Receive Queue Parameters register 3 | R/W | 0x0000_0000 | |
| 0x2_4C10 | RQPRM4*—Receive Queue Parameters register 4 | R/W | 0x0000_0000 | |
| 0x2_4C14 | RQPRM5*—Receive Queue Parameters register 5 | R/W | 0x0000_0000 | |
| 0x2_4C18 | RQPRM6*—Receive Queue Parameters register 6 | R/W | 0x0000_0000 | |
| 0x2_4C1C | RQPRM7*—Receive Queue Parameters register 7 | R/W | 0x0000_0000 | |
| 0x2_4C20–<br>0x2_4C40 | Reserved | — | — | — |
| 0x2_4C44 | RFBPTR0*—Last Free RxBD pointer for ring 0 | R/W | 0x0000_0000 | 14.5.3.10.2/14-121 |
| 0x2_4C48 | Reserved | — | — | — |
| 0x2_4C4C | RFBPTR1*—Last Free RxBD pointer for ring 1 | R/W | 0x0000_0000 | 14.5.3.10.2/14-121 |
| 0x2_4C50 | Reserved | — | — | — |
| 0x2_4C54 | RFBPTR2*—Last Free RxBD pointer for ring 2 | R/W | 0x0000_0000 | 14.5.3.10.2/14-121 |
| 0x2_4C58 | Reserved | — | — | — |
| 0x2_4C5C | RFBPTR3*—Last Free RxBD pointer for ring 3 | R/W | 0x0000_0000 | 14.5.3.10.2/14-121 |
| 0x2_4C60 | Reserved | — | — | — |
| 0x2_4C64 | RFBPTR4*—Last Free RxBD pointer for ring 4 | R/W | 0x0000_0000 | 14.5.3.10.2/14-121 |
| 0x2_4C68 | Reserved | — | — | — |
| 0x2_4C6C | RFBPTR5*—Last Free RxBD pointer for ring 5 | R/W | 0x0000_0000 | 14.5.3.10.2/14-121 |
| 0x2_4C70 | Reserved | — | — | — |
| 0x2_4C74 | RFBPTR6*—Last Free RxBD pointer for ring 6 | R/W | 0x0000_0000 | 14.5.3.10.2/14-121 |
| 0x2_4C78 | Reserved | — | — | — |
| 0x2_4C7C | RFBPTR7*—Last Free RxBD pointer for ring 7 | R/W | 0x0000_0000 | 14.5.3.10.2/14-121 |
| **eTSEC Future Expansion Space** | | | | |
| 0x2_4CC0<br>–<br>0x2_4D94 | Reserved | — | — | — |
| **eTSEC IEEE 1588 Registers** | | | | |
| 0x2_4E00 | TMR_CTRL* - Timer control register | R/W | 0x0001_0001 | 14.5.3.11.1/14-122 |
| 0x2_4E04 | TMR_TEVENT* - time stamp event register | w1c | 0x0000_0000 | 14.5.3.11.2/14-124 |

**Table 14-4. Module Memory Map (continued)**

| eTSEC1 Offset | Name[1] | Access [2] | Reset | Section/Page |
|---|---|---|---|---|
| 0x2_4E08 | TMR_TEMASK* - Timer event mask register | R/W | 0x0000_0000 | 14.5.3.11.3/14-125 |
| 0x2_4E0C | TMR_PEVENT* - time stamp event register | R/W | 0x0000_0000 | 14.5.3.11.4/14-126 |
| 0x2_4E10 | TMR_PEMASK* - Timer event mask register | R/W | 0x0000_0000 | 14.5.3.11.5/14-127 |
| 0x2_4E14 | TMR_STAT* - time stamp status register | R/W | 0x0000_0000 | 14.5.3.11.6/14-128 |
| 0x2_4E18 | TMR_CNT_H* - timer counter high register | R/W | 0x0000_0000 | 14.5.3.11.7/14-128 |
| 0x2_4E1C | TMR_CNT_L* - timer counter low register | R/W | 0x0000_0000 | 14.5.3.11.7/14-128 |
| 0x2_4E20 | TMR_ADD* - Timer drift compensation addend register | R/W | 0x0000_0000 | 14.5.3.11.8/14-129 |
| 0x2_4E24 | TMR_ACC* - Timer accumulator register | R/W | 0x0000_0000 | 14.5.3.11.9/14-130 |
| 0x2_4E28 | TMR_PRSC* -Timer prescale | R/W | 0x0000_0002 | 14.5.3.11.10/14-130 |
| 0x2_4E2C | Reserved | — | — | — |
| 0x2_4E30 | TMROFF_H* - Timer offset high | R/W | 0x0000_0000 | 14.5.3.11.11/14-131 |
| 0x2_4E34 | TMROFF_L* - Timer offset low | R/W | 0x0000_0000 | 14.5.3.11.11/14-131 |
| 0x2_4E40 | TMR_ALARM1_H* - Timer alarm 1 high register | R/W | 0xFFFF_FFFF | 14.5.3.11.12/14-131 |
| 0x2_4E44 | TMR_ALARM1_L* - Timer alarm 1 high register | R/W | 0xFFFF_FFFF | |
| 0x2_4E48 | TMR_ALARM2_H* - Timer alarm 2 high register | R/W | 0xFFFF_FFFF | |
| 0x2_4E4C | TMR_ALARM2_L* - Timer alarm 2 high register | R/W | 0xFFFF_FFFF | |
| 0x2_4E50–0x2_4E7C | Reserved | — | — | — |
| 0x2_4E80 | TMR_FIPER1* - Timer fixed period interval | R/W | 0xFFFF_FFFF | 14.5.3.11.13/14-132 |
| 0x2_4E84 | TMR_FIPER2* - Timer fixed period interval | R/W | 0xFFFF_FFFF | |
| 0x2_4E88 | TMR_FIPER*3 - Timer fixed period interval | R/W | 0xFFFF_FFFF | |
| 0x2_4EA0 | TMR_ETTS1_H* - Time stamp of general purpose external trigger | R/W | 0x0000_0000 | 14.5.3.11.14/14-133 |
| 0x2_4EA4 | TMR_ETTS1_L* - Time stamp of general purpose external trigger | R/W | 0x0000_0000 | |
| 0x2_4EA8 | TMR_ETTS2_H* - Time stamp of general purpose external trigger | R/W | 0x0000_0000 | |
| 0x2_4EAC | TMR_ETTS2_L* - Time stamp of general purpose external trigger | R/W | 0x0000_0000 | |
| 0x2_4EB0 – 0x2_4FFF | Reserved | — | — | |
| **Other eTSECs** | | | | |
| 0x2_6000–0x2_6FFF | eTSEC3 REGISTERS[4] | | | |

[1] Registers denoted * are new to the enhanced TSEC and not supported by PowerQUICC III TSECs.

[2] Key: R = read only, WO = write only, R/W = read and write, LH = latches high, SC = self-clearing.

[3] Cleared on read.

[4] eTSEC3 has the same memory-mapped registers that are described for eTSEC1 from 0x 2_4000 to 0x2_4FFF, except the offsets are from 0x 2_6000 to 0x2_6FFF.

## 14.5.3 Memory-Mapped Register Descriptions

This section provides a detailed description of all the eTSEC registers. Because all of the eTSEC registers are 32 bits wide, only 32-bit register accesses are supported.

### 14.5.3.1 eTSEC General Control and Status Registers

This section describes general control and status registers used for both transmitting and receiving Ethernet frames. All of the registers are 32 bits wide.

#### 14.5.3.1.1 Controller ID Register (TSEC_ID)

The controller ID register (TSEC_ID) is a read-only register. The TSEC_ID register is used to identify the eTSEC block and revision.

Offset eTSEC1:0x2_4000;
eTSEC3:0x2_6000

Access: Read only

| | 0–15 | 16–23 | 24–31 |
|---|---|---|---|
| R | TSEC_ID | TSEC_REV_MJ | TSEC_REV_MN |
| W | | | |
| Reset | 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 |

**Figure 14-2. TSEC_ID Register**

Table 14-10 describes the fields of the TSEC_ID register.

**Table 14-5. TSEC_ID Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–15 | TSEC_ID | Value identifying the eTSEC (10/100/1000 Ethernet MAC).<br>0124 Unique identifier for eTSEC with 8 Rx and 8 Tx BD rings. |
| 16–23 | TSEC_REV_MJ | Value identifies the major revision of the eTSEC.<br>00 Initial revision |
| 24–31 | TSEC_REV_MN | Value identifies the minor revision of the eTSEC. |

#### 14.5.3.1.2 Controller ID Register (TSEC_ID2)

The controller ID register (TSEC_ID2) is a read-only register. The TSEC_ID2 register is used to identify the eTSEC block configuration.

Offset eTSEC1:0x2_4004;
eTSEC3:0x2_6004

Access: Read only

| | 0 | | | | | | | | | 9 | 10 | | | | | 15 | 16 | | | | | | | 23 | 24 | | | | | | | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | — | | | | | | | | | | TSEC_INT | | | | | | — | | | | | | | | TSEC_CFG | | | | | | | |
| W | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |

**Figure 14-3. TSEC_ID2 Register**

Table 14-6 describes the fields of the TSEC_ID2 register.

**Table 14-6. TSEC_ID2 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–9 | — | Reserved |
| 10–15 | TSEC_INT | Interface mode support. See Table 14-7 for settings. |
| 16–23 | — | Reserved |
| 24–31 | TSEC_CFG | Value identifies configuration options of the eTSEC.<br>00 eTSEC multiple ring, Rx TOE, Filer and Tx TOE supports are off<br>F0 eTSEC multiple ring, Rx TOE, Filer and Tx TOE supports are on<br>30 eTSEC multiple ring support is OFF and Rx TOE, Filer and Tx TOE supports are on<br>50 eTSEC multiple ring and filer supports are OFF and Rx TOE and Tx TOE supports are on |

Table 14-7 describes the field settings for TSEC_ID2[TSEC_INT].

**Table 14-7. TSEC_ID2[TSEC_INT] Field Settings**

| Bit | Mode |
|---|---|
| 10 | 0 Ethernet mode not supported<br>1 Ethernet mode supported |
| 11 | 0 FIFO mode not supported<br>1 FIFO mode supported |
| 12 | Reserved |
| 13 | 0 Can be configured to run in FIFO 8-bit mode<br>1 FIFO 8-bit mode off |
| 14 | 0 Can be configured to run in Ethernet normal/full mode<br>1 Ethernet normal/full mode off |
| 15 | 0 Can be configured to run in Ethernet reduced mode<br>1 Ethernet reduced mode off |

#### 14.5.3.1.3 Interrupt Event Register (IEVENT)

Interrupt events cause bits in the IEVENT register to be set. Software may poll this register at any time to check for pending interrupts. If an event occurs and its corresponding enable bit is set in the interrupt mask register (IMASK), the event also causes a hardware interrupt at the PIC. A bit in the interrupt event register is cleared by writing a 1 to that bit position. A write of 0 has no effect.

Each eTSEC can issue three kinds of hardware interrupt to the PIC:

1. Transmit data frame interrupts—Issued whenever bits TXB or TXF of IEVENT are set to 1 and either transmit interrupt coalescing is disabled or the interrupt coalescing thresholds have been met for TXF. To negate this hardware interrupt, software must clear both TXB and TXF bits.
2. Receive data frame interrupts—Issued whenever bits RXB or RXF of IEVENT are set to 1 and either receive interrupt coalescing is disabled or the interrupt coalescing thresholds have been met for RXF. To negate this hardware interrupt, software must clear both RXB and RXF bits.
3. Error, diagnostic, and special interrupts—Issued whenever bits MAG, GTSC, GRSC, TXC, RXC, BABR, BABT, LC, CRL, FGPI, FIR, FIQ, DPE, PERR, EBERR, TXE, XFUN, BSY, MSRO, MMRD, or MMRW of IEVENT are set to 1. Software must clear all of these bits to negate an error/diagnostic/special hardware interrupt.
   - Magic Packet reception event is: MAG
   - Operational diagnostics are events on: GTSC, GRSC, TXC, and RXC
   - Interrupts resulting from errors/problems detected in the network or transceiver are: BABR, BABT, LC, and CRL
   - Interrupts resulting from internal or combination errors are: FIR, FIQ, DPE, PERR, EBERR, TXE, XFUN, and BSY
   - Special function interrupts are: FGPI, MSRO, MMRD, and MMRW

Some of the error interrupts are independently counted in the MIB block counters. Software may choose to mask off these interrupts because these errors are visible to network management through the MIB counters.

Figure 14-4 describes the definition for the IEVENT register.

Offset eTSEC1:0x2_4010;
eTSEC3:0x2_6010

Access: w1c

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | BABR | RXC | BSY | EBERR | — | MSRO | GTSC | BABT | TXC | TXE | TXB | TXF | — | LC | CRL | XFUN |
| W | w1c | w1c | w1c | w1c | | w1c | w1c | w1c | w1c | w1c | w1c | w1c | | w1c | w1c | w1c |
| Reset | All zeros | | | | | | | | | | | | | | | |

| | 16 | 17 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | RXB | — | | MAG | MMRD | MMWR | GRSC | RXF | — | | FGPI | FIR | FIQ | DPE | PERR |
| W | w1c | | | w1c | w1c | w1c | w1c | w1c | | | w1c | w1c | w1c | w1c | w1c |
| Reset | All zeros | | | | | | | | | | | | | | |

**Figure 14-4. IEVENT Register Definition**

Table 14-8 describes the fields of the IEVENT register.

**Table 14-8. IEVENT Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | BABR | Babbling receive error. This bit indicates that a frame was received with length in excess of the MAC's maximum frame length register while MACCFG2[Huge Frame] is set.<br>0 Excessive frame not received.<br>1 Excessive frame received. |
| 1 | RXC | Receive control interrupt. A control frame was received while MACCFG1[Rx_Flow] is set. As soon as the transmitter finishes sending the current frame, a pause operation is performed.<br>0 Control frame not received.<br>1 Control frame received. |
| 2 | BSY | Busy condition interrupt. Indicates that a frame was received and discarded due to a lack of buffers.<br>0 No frame received and discarded.<br>1 Frame received and discarded. |
| 3 | EBERR | Internal bus error. This bit indicates that a system bus error occurred while a DMA transaction was underway. As a result, transferred data is expected to be partially or completely invalid.<br>0 No system bus error occurred.<br>1 System bus error occurred. |
| 4 | — | Reserved |
| 5 | MSRO | MIB counter overflow. This interrupt is asserted if the count for one of the MIB counters has exceeded the size of its register.<br>0 MIB count not exceeding its register size.<br>1 MIB count exceeds its register size. |
| 6 | GTSC | Graceful transmit stop complete. This interrupt is asserted for one of two reasons. Graceful stop means that the transmitter is put into a pause state after completion of the frame currently being transmitted.<br>• A graceful stop, which was initiated by setting DMACTRL[GTS], is now complete.<br>• A transmission of a flow control PAUSE frame, which was initiated by setting TCTRL[TFC_PAUSE], is now complete.<br>0 No graceful stop interrupt.<br>1 Graceful stop requested. |
| 7 | BABT | Babbling transmit error. This bit indicates that the transmitted frame length has exceeded the value in the MAC's maximum frame length register and MACCFG2[Huge Frame] is cleared. Frame truncation occurs when this condition occurs.<br>0 Transmitted frame length not exceeding maximum frame length.<br>1 Transmitted frame length exceeding maximum frame length when MACCFG2[Huge Frame] = 0. |
| 8 | TXC | Transmit control interrupt. This bit indicates that a control frame was transmitted.<br>0 Control frame not transmitted.<br>1 Control frame transmitted. |
| 9 | TXE | Transmit error. This bit indicates that an error occurred on the transmitted channel that has caused TSTAT[THLT] to be set by the eTSEC. This bit is set whenever any transmit error occurs that causes the transmitter to halt (EBERR, LC, CRL, XFUN).<br>0 No transmit channel error occurred.<br>1 Transmit channel error occurred. |
| 10 | TXB | Transmit buffer. This bit indicates that a transmit buffer descriptor was updated whose I (interrupt) bit was set in its status word and was not the last buffer descriptor of the frame.<br>0 No transmit buffer descriptor updated.<br>1 Transmit buffer descriptor updated. |

**Table 14-8. IEVENT Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 11 | TXF | Transmit frame interrupt. This bit indicates that a frame was transmitted and that the last corresponding transmit buffer descriptor (TxBD) was updated. This only occurs if the I (interrupt) bit in the status word of the buffer descriptor is set. The specific transmit queue that was updated has its TXF bit set in TSTAT.<br>0 No frame transmitted/TxBD not updated.<br>1 Frame transmitted/TxBD updated. |
| 12 | — | Reserved |
| 13 | LC | Late collision. This bit indicates that a collision occurred beyond the collision window (slot time) in half-duplex mode. The frame is truncated with a bad CRC and the remainder of the frame is discarded.<br>0 No late collision occurred.<br>1 Late collision occurred. |
| 14 | CRL | Collision retry limit. This bit indicates that the number of successive transmission collisions has exceeded the MAC’s half-duplex register’s retransmission maximum count (HAFDUP[Retransmission Maximum]). The frame is discarded without being transmitted and the queue halts (TSTAT[THLT*n*] set to 1). This only occurs while in half-duplex mode.<br>0 Successive transmission collisions do not exceed maximum.<br>1 Successive transmission collisions exceed maximum. |
| 15 | XFUN | Transmit FIFO underrun. This bit indicates that the transmit FIFO became empty before the complete frame was transmitted.<br>0 Transmit FIFO not underrun.<br>1 Transmit FIFO underrun. |
| 16 | RXB | Receive buffer. This bit indicates that a receive buffer descriptor was updated which had the I (Interrupt) bit set in its status word and was not the last buffer descriptor of the frame.<br>0 Receive buffer descriptor not updated.<br>1 Receiver buffer descriptor updated. |
| 17–19 | — | Reserved |
| 20 | MAG | Magic Packet detected when the eTSEC is in Magic Packet detection mode (MACCFG2[MPEN] = 1).<br>0 No Magic Packet received, or Magic Packet mode was not enabled.<br>1 A Magic Packet was received while in Magic Packet mode. MACCFG2[MPEN] is also cleared upon receiving the Magic Packet. |
| 21 | MMRD | MII management read completion<br>0 MII management read not issued or in process.<br>1 MII management read completed that was initiated by a user through the MII Scan or Read cycle command. |
| 22 | MMWR | MII management write completion<br>0 MII management write not issued or in process.<br>1 MII management write completed that was initiated by a user write to the MIIMCON register. |
| 23 | GRSC | Graceful receive stop complete. This interrupt is asserted if a graceful receive stop is completed. It allows the user to know if the system has completed the stop and it is safe to write to receive registers (status, control or configuration registers) that are used by the system during normal operation.<br>0 Graceful stop not completed.<br>1 Graceful stop completed. |
| 24 | RXF | Receive frame interrupt. This bit indicates that a frame was received and the last receive buffer descriptor (RxBD) in that frame was updated. This occurs either if the I (interrupt) bit in the buffer descriptor status word is set, or an overrun error occurs. The specific receive queue that was updated has its RXF bit set in RSTAT.<br>0 Frame not received.<br>1 Frame received. |
| 25–26 | — | Reserved |

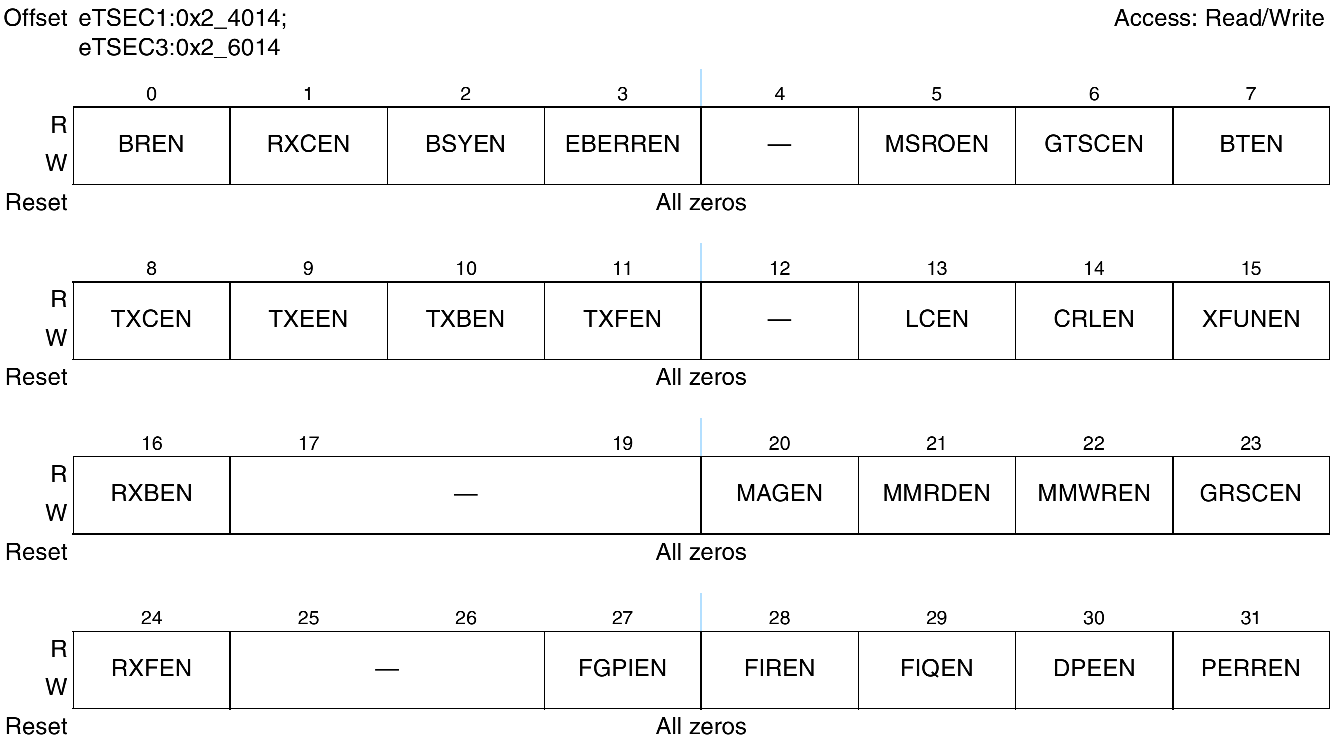
Table 14-8. IEVENT Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 27 | FGPI | Filer generated general purpose interrupt on a set of filer rule match. This bit will be set upon reception of a frame that matches a GPI rule sequence that is specified in the filer. It is synchronized with the setting of RXF.<br>0 No filer generated interrupt has occurred.<br>1 The filer has accepted a frame via a matching rule that the RQFCR[GPI] bit set. |
| 28 | FIR | The receive queue filer result is invalid, either because not enough time between frames was available to find a matching rule, or no entry in the filer table could be matched.<br>0 Receive queue filer reached a definite result; however, bit FIQ may still be set if a frame was filed to a disabled RxBD ring.<br>1 Receive queue filer was unable to reach a definite result. In this case, bit FIQ is also set if no entry in the filer table could provide a rule match. |
| 29 | FIQ | Filed frame to invalid receive queue. This bit indicates that either the receive queue filer chose to DMA a received frame to a disabled RxBD ring, or that no rule in the filer table could be matched.<br>0 Received frames filed to valid queues or rejected. Note that a frame may be rejected if the filer has insufficient time to reach a conclusive result between frames, in which case bit FIR is set.<br>1 Received frames filed to RxBD rings that are not enabled. The frame is discarded. If bit FIR is also set this indicates that the filer exhausted all of its table entries without a rule match. |
| 30 | DPE | Internal data parity error. This bit indicates that the eTSEC has detected a parity error on its stored data, which is likely to compromise the validity of recently transferred frames.<br>0 No parity errors detected.<br>1 Data held in the FIFO or filer arrays is expected to be corrupted due to a parity error. |
| 31 | PERR | Receive frame parse error for TCP/IP off-load. This bit indicates that a received frame could not be parsed unambiguously, due to encapsulated header type fields contradicting each other.<br>0 Received frame parsed successfully.<br>1 Received frame parse revealed header inconsistencies. |

#### 14.5.3.1.4 Interrupt Mask Register (IMASK)

The interrupt mask register provides control over which possible interrupt events in the IEVENT register are permitted to participate in generating hardware interrupts to the PIC. All implemented bits in this register are R/W and cleared upon a hardware reset. If the corresponding bits in both the IEVENT and IMASK registers are set, the PIC receives an interrupt (for each eTSEC these are grouped into transmit, receive, and error/diagnostic interrupts). The interrupt signal remains asserted until either the IEVENT bit is cleared, by writing a 1 to it, or by writing a 0 to the corresponding IMASK bit.

Figure 14-5 describes the IMASK register.



Offset eTSEC1:0x2_4014;
eTSEC3:0x2_6014

Access: Read/Write

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| R<br>W | BREN | RXCEN | BSYEN | EBERREN | — | MSROEN | GTSCEN | BTEN |
| Reset | All zeros | | | | | | | |

| | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|
| R<br>W | TXCEN | TXEEN | TXBEN | TXFEN | — | LCEN | CRLEN | XFUNEN |
| Reset | All zeros | | | | | | | |

| | 16 | 17 | | 19 | 20 | 21 | 22 | 23 |
|---|---|---|---|---|---|---|---|---|
| R<br>W | RXBEN | — | | | MAGEN | MMRDEN | MMWREN | GRSCEN |
| Reset | All zeros | | | | | | | |

| | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|
| R<br>W | RXFEN | — | | FGPIEN | FIREN | FIQEN | DPEEN | PERREN |
| Reset | All zeros | | | | | | | |

**Figure 14-5. IMASK Register Definition**

Table 14-9 describes the fields of the IMASK register.

**Table 14-9. IMASK Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | BREN | Babbling receiver interrupt enable |
| 1 | RXCEN | Receive control interrupt enable |
| 2 | BSYEN | Busy interrupt enable |
| 3 | EBERREN | Ethernet controller bus error enable |
| 4 | — | Reserved |
| 5 | MSROEN | MIB counter overflow interrupt enable |
| 6 | GTSCEN | Graceful transmit stop complete interrupt enable |
| 7 | BTEN | Babbling transmitter interrupt enable |
| 8 | TXCEN | Transmit control interrupt enable |
| 9 | TXEEN | Transmit error interrupt enable |
| 10 | TXBEN | Transmit buffer interrupt enable |
| 11 | TXFEN | Transmit frame interrupt enable |
| 12 | — | Reserved |

**Table 14-9. IMASK Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 13 | LCEN | Late collision enable |
| 14 | CRLEN | Collision retry limit enable |
| 15 | XFUNEN | Transmit FIFO underrun enable |
| 16 | RXBEN | Receive buffer interrupt enable |
| 17–19 | — | Reserved |
| 20 | MAGEN | Magic packet received interrupt enable |
| 21 | MMRDEN | MII management read completion interrupt enable |
| 22 | MMWREN | MII management write completion interrupt enable |
| 23 | GRSCEN | Graceful receive stop complete interrupt enable |
| 24 | RXFEN | Receive frame interrupt enable |
| 25–26 | — | Reserved |
| 27 | FGPIEN | Filer general purpose interrupt enable |
| 28 | FIREN | Filer invalid result interrupt enable |
| 29 | FIQEN | Filed frame to invalid queue interrupt enable |
| 30 | DPEEN | Data parity error interrupt enable |
| 31 | PERREN | Receive frame parse error enable |

##### 14.5.3.1.5 Error Disabled Register (EDIS)

Figure 14-6 describes the definition for the EDIS register. The error disabled register allows the user to disable an error interruption, possibly to avoid spurious error indications external to the eTSECs.

Offset eTSEC1:0x2_4018;
eTSEC3:0x2_6018

Access: Read/Write

| | 0 1 | 2 | 3 | 4 6 | 7 | 8 | 9 | 10 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | BSYDIS | EBERRDIS | — | BABTDIS | — | TXEDIS | — | LCDIS | CRLDIS | XFUNDIS |
| Reset | All zeros | | | | | | | | | | |

| | 16 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|
| R<br>W | — | FIRDIS | FIQDIS | DPEDIS | PERRDIS |
| Reset | All zeros | | | | |

**Figure 14-6. EDIS Register Definition**

Table 14-10 describes the fields of the EDIS register.

Table 14-10. EDIS Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–1 | — | Reserved |
| 2 | BSYDIS | Busy disable.<br>0 Allow eTSEC to report IEVENT[BSY] status and halt buffer descriptor queue if BSY condition occurs.<br>1 Do not set IEVENT[BSY] and do not halt buffer descriptor queue if BSY condition occurs. |
| 3 | EBERRDIS | Ethernet controller bus error disable.<br>0 Allow eTSEC to report IEVENT[EBERR] status and halt buffer descriptor queue if EBERR condition occurs.<br>1 Do not set IEVENT[EBERR] and do not halt buffer descriptor queue if EBERR condition occurs. |
| 4–6 | — | Reserved |
| 7 | BABTDIS | Babbling transmit error disable.<br>0 Allow eTSEC to report IEVENT[BABT] status and set the buffer descriptor TR field.<br>1 Do not set IEVENT[BABT] nor the buffer descriptor TR field. |
| 8 | — | Reserved |
| 9 | TXEDIS | Transmit error disable.<br>0 Allow eTSEC to report IEVENT[TXE] status.<br>1 Do not set IEVENT[TXE] if TXE condition occurs. |
| 10–12 | — | Reserved |
| 13 | LCDIS | Late collision disable.<br>0 Allow eTSEC to report IEVENT[LC] status, set the buffer descriptor LC field, and halt buffer descriptor queue if LC condition occurs.<br>1 Do not set IEVENT[LC] nor the buffer descriptor LC field, and do not halt buffer descriptor queue if LC condition occurs. |
| 14 | CRLDIS | Collision retry limit disable.<br>0 Allow eTSEC to report IEVENT[CRL] status, set the buffer descriptor RL field, and halt buffer descriptor queue if CRL condition occurs.<br>1 Do not set IEVENT[CRL] nor the buffer descriptor RL field, and do not halt buffer descriptor queue if CRL condition occurs. |
| 15 | XFUNDIS | Transmit FIFO underrun disable.<br>0 Allow eTSEC to report IEVENT[XFUN] status, set the buffer descriptor UN field, and halt buffer descriptor queue if XFUN condition occurs.<br>1 Do not set IEVENT[XFUN] nor the buffer descriptor UN field, and do not halt buffer descriptor queue if XFUN condition occurs. |
| 16–27 | — | Reserved |
| 28 | FIRDIS | Filer invalid result error disable.<br>0 Allow eTSEC to report IEVENT[FIR] status.<br>1 Do not set IEVENT[FIR] if eTSEC fails to reach a definite filer result when attempting to file a received frame, but discard the frame silently. |
| 29 | FIQDIS | Filed frame to invalid queue error disable.<br>0 Allow eTSEC to report IEVENT[FIQ] status.<br>1 Do not set IEVENT[FIQ] if eTSEC attempts to file a received frame to an invalid (disabled) RxBD ring, but discard the frame silently. |

**Table 14-10. EDIS Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 30 | DPEDIS | Data parity error disable.<br>0 Allow eTSEC to report IEVENT[DPE] status.<br>1 Do not set IEVENT[DPE] if a parity error occurs in eTSEC's FIFO or filer arrays. |
| 31 | PERRDIS | Receive frame parse error disable.<br>0 Allow eTSEC to report IEVENT[PERR] status.<br>1 Do not set IEVENT[PERR] if a parse error occurs on a received frame. |

### 14.5.3.1.6 Ethernet Control Register (ECNTRL)

ECNTRL is a register writable by the user to reset, configure, and initialize the eTSEC. Note that the FIFM, GMIIM, TBIM, RPM, and RMM fields are read-only, having been set after sampling signals at power-on-reset.

Figure 14-7 describes the definition for the ECNTRL register.

Offset eTSEC1:0x2_4020;
eTSEC3:0x2_6020

Access: Mixed

| | 0–15 |
|---|---|
| R | — |
| W | |
| Reset | All zeros |

| | 16 | 17 | 18 | 19 | 20–24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | FIFM | CLRCNT | AUTOZ | STEN | — | GMIIM | TBIM | RPM | R100M | RMM | SGMIIM | — |
| W | | | | | | | | | | | | |
| Reset | All zeros | | | | | | | | | | | |

**Figure 14-7. ECNTRL Register Definition**

Table 14-11 describes the fields of the ECNTRL register.

**Table 14-11. ECNTRL Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–15 | — | Reserved |
| 16 | FIFM | FIFO mode enable. If this bit is set, 8-bit FIFO interface mode is enabled. This bit can be pin configured at reset to set or clear. See Section 4.4.3, "Power-On Reset Configuration."<br>0 Interface to external signals through the Ethernet MAC.<br>1 Interface to external signals through the 8-bit FIFO interface, bypassing the Ethernet MAC. Frame parsing in this mode automatically assumes that IP packets are being received and transmitted. See FIFOCFG register for configuration of the FIFO interface. |
| 17 | CLRCNT | Clear all statistics counters and carry registers.<br>0 Allow MIB counters to continue to increment and keep any overflow indicators.<br>1 Reset all MIB counters and CAR1 and CAR2.<br>This bit is self-resetting. |

Table 14-11. ECNTRL Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 18 | AUTOZ | Automatically zero MIB counter values and carry registers.<br>0 The user must write the addressed counter zero after a host read.<br>1 The addressed counter value is automatically cleared to zero after a host read.<br>This is a steady state signal and must be set prior to enabling the Ethernet controller and must not be changed without proper care. |
| 19 | STEN | MIB counter statistics enabled.<br>0 Statistics not enabled<br>1 Enables internal counters to update<br>This is a steady state signal and must be set prior to enabling the Ethernet controller and must not be changed without proper care. |
| 20–24 | — | Reserved |
| 25 | GMIIM | GMII interface mode. If this bit is set, a PHY with a GMII or RGMII interface is expected to be connected. If cleared, a PHY with an MII or RMII interface is expected. The user should then set MACCFG2[I/F Mode] accordingly. The state of this status bit is defined during power-on reset. See Section 4.4.3, "Power-On Reset Configuration."<br>0 MII or RMII mode interface expected<br>1 GMII or RGMII mode interface expected |
| 26 | TBIM | Ten-bit interface mode. If this bit is set, ten-bit interface mode is enabled. This bit can be pin-configured at reset to set or clear. See Section 4.4.3, "Power-On Reset Configuration."<br>0 GMII or MII or RMII mode interface<br>1 TBI mode interface |
| 27 | RPM | Reduced-pin mode for Gigabit interfaces. If this bit is set, a reduced-pin interface is expected on either Ethernet and FIFO interfaces. RPM and RMM are never set together. This register can be pin-configured at reset to 0 or 1. See Section 4.4.3, "Power-On Reset Configuration."<br>0 GMII or MII or TBI in non-reduced-pin mode configuration<br>1 RGMII or RTBI reduced-pin mode<br>FIFO configured for 8-bit operation |
| 28 | R100M | RGMII/RMII 100 mode. This bit is ignored unless SGMIIM, RPM or RMM are set and MACCFG2[I/F Mode] is assigned to 10/100 (01).<br>0 RGMII is in 10 Mbps mode<br>RMII is in 10 Mbps mode, and every 10th RMII Reference clock is used to transfer data<br>SGMII is in 10 Mbps mode, and every 100th SGMII Reference clock is used to transfer data<br>1 RGMII is in 100 Mbps mode<br>RMII is in 100 Mbps mode, and data is transferred on every Reference clock<br>SGMII is in 100 Mbps mode, and every 10th SGMII Reference clock is used to transfer data<br>This bit must be cleared for 1-Gbps SGMII operation. |
| 29 | RMM | Reduced-pin mode for 10/100 interfaces. If this bit is set, an RMII pin interface is expected. RMM must be 0 if RPM = 1. This register can be pin-configured at reset to 0 or 1. See Section 4.4.3, "Power-On Reset Configuration."<br>0 Non-RMII interface mode<br>1 RMII interface mode |

Table 14-11. ECNTRL Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 30 | SGMIIM | Serial GMII mode. If this bit is set, a SGMII pin interface is expected to be connected via an on chip SerDes.<br>This register can be pin-configured at reset to 0 or 1. See Section 4.4.3, "Power-On Reset Configuration."<br>0 SGMII mode disabled. eTSEC connected via a parallel interface.<br>1 SGMII mode enabled. |
| 31 | — | Reserved |

The different interface configurations indicated by registers ECNTRL and MACCFG2 are summarized in Table 14-12.

Table 14-12. eTSEC Interface Configurations

| Interface Mode | ECNTRL Field | | | | | | | MACCFG2 Field |
|---|---|---|---|---|---|---|---|---|
| | FIFM | GMIIM | TBIM | RPM | R100M | RMM | SGMIIM | I/F Mode |
| FIFO 8-bits | 1 | 0 | 0 | 1 | 0 | 0 | 0 | — |
| TBI 1Gbps | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 10 |
| RTBI 1Gbps | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 10 |
| GMII 1Gbps[1] | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 10 |
| RGMII 1Gbps | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 10 |
| RGMII 100 Mbps | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 01 |
| RGMII 10 Mbps | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 01 |
| MII 10/100 Mbps | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 01 |
| RMII 100 Mbps | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 01 |
| RMII 10 Mbps | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 01 |
| SGMII 1 Gbps | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 10 |
| SGMII 100 Mbps | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 01 |
| SGMII 10 Mbps | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 01 |

[1] See MII 10/100 Mbps mode for GMII 10/100 Mbps 'fall-back' mode.

#### 14.5.3.1.7 Pause Time Value Register (PTV)

PTV is a 32-bit register written by the user to store the pause duration used when the eTSEC initiates an IEEE 802.3 PAUSE control frame through TCTRL[TFC_PAUSE]. The low-order 16 bits (PT) represent the pause time and the high-order 16 bits (PTE) represent the extended pause control parameter. The pause time is measured in units of *pause_quanta*, equal to 512 bit times. The pause time can range from 0 to 65,535 *pause_quanta*, or 0 to 33,553,920 bit times. See Section 14.6.3.9, "Flow Control," for additional details. Figure 14-8 describes the definition for the PTV register.

Offset eTSEC1:0x2_4028;
eTSEC3:0x2_6028

Access: Read/Write

| | 0–15 | 16–31 |
|---|---|---|
| R<br>W | PTE | PT |
| Reset | All zeros | |

**Figure 14-8. PTV Register Definition**

Table 14-13 describes the fields of the PTV register.

**Table 14-13. PTV Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–15 | PTE | Extended pause control. This field allows software to add a 16-bit additional control parameter into the PAUSE frame to be sent when TCTRL[TFC_PAUSE] is set. Note that current IEEE 802.3 PAUSE frame format requires this parameter to be cleared. |
| 16–31 | PT | Pause time value. Represents the 16-bit pause quanta (that is, 512 bit times). This pause value is used as part of the PAUSE frame to be sent when TCTRL[TFC_PAUSE] is set. See Section 14.6.3.9, “Flow Control,” on page 14-170 for more information. |

### 14.5.3.1.8 DMA Control Register (DMACTRL)

DMACTRL is writable by the user to configure the DMA block. Figure 14-9 describes the definition for the DMACTRL register.

Offset eTSEC1:0x2_402C;
eTSEC3:0x2_602C

Access: Read/Write

| | 0–15 | 16 | 17–23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | LE | — | TDSEN | TBDSEN | — | GRS | GTS | TOD | WWR | WOP |
| Reset | All zeros | | | | | | | | | | |

**Figure 14-9. DMACTRL Register**

Table 14-14 describes the fields of the DMACTRL register.

**Table 14-14. DMACTRL Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–15 | — | Reserved |
| 16 | LE | Little-endian descriptor mode enable. This bit controls both the reading and writing of descriptors; data buffers are always transferred in network byte order.<br>0 RxBDs and TxBDs are interpreted with big-endian byte ordering, as shown in Section 14.6.8.1, “Data Buffer Descriptors.”<br>1 RxBDs and TxBDs are interpreted with little-endian byte ordering. That is, the 16 bits of flags are considered a complete half-word unit, the buffer length is considered another complete half-word unit, and the buffer pointer is considered a complete word unit. |
| 17–23 | — | Reserved |

Table 14-14. DMACTRL Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 24 | TDSEN | Tx Data snoop enable.<br>0 Disables snooping of all transmit frames from memory.<br>1 Enables snooping of all transmit frames from memory. |
| 25 | TBDSEN | TxBD snoop enable.<br>0 Disables snooping of all transmit BD memory accesses.<br>1 Enables snooping of all transmit BD memory accesses. |
| 26 | — | Reserved |
| 27 | GRS | Graceful receive stop. If this bit is set, the Ethernet controller stops receiving frames following completion of the frame currently being received. (That is, after a valid end of frame was received). The contents of the Rx FIFO are then written to memory, and the IEVENT[GRSC] is set to indicate that all current receive buffers have been closed. Because the receive enable bit of the MAC may still be set, the MAC may continue to receive but the eTSEC ignores the receive data until GRS is cleared. If this bit is cleared, the eTSEC scans the input data stream for the start of a new frame (preamble sequence and start of frame delimiter) and the first valid frame received uses the next RxBD.<br>If GRS is set, the user must monitor the graceful receive stop complete (GRSC) bit in the IEVENT register to insure that the graceful receive stop was completed. The user can then clear IEVENT[GRSC] and can write to receive registers that are accessible to both user and the eTSEC hardware without fear of conflict.<br>0 eTSEC scans input data stream for valid frame.<br>1 eTSEC stops receiving frames following completion of current frame. |
| 28 | GTS | Graceful transmit stop. If this bit is set, the Ethernet controller stops transmission after all frames that are currently in the Tx FIFO or scheduled have been transmitted, and the GTSC interrupt in the IEVENT register is asserted. A frame that has started reading buffer descriptors or data from memory is read to completion and transmitted before the GTSC interrupt occurs. However, if no frame has been scheduled for transmission and the Tx FIFO is empty, the GTSC interrupt is asserted immediately. Once transmission has completed, clearing GTS “restart” transmit.<br>0 Controller continues.<br>1 Controller stops transmission after completion of current frame. |
| 29 | TOD | Transmit on demand for TxBD ring 0. This bit is applicable only to the transmitter, and requires both TCTRL[TXSCHED] = 00 and DMACTRL[WOP] = 0. If 1 is written to this bit, the eTSEC immediately begins fetching the next TxBD from ring 0, avoiding waiting the normal polling time to check the TxBD’s R bit. This bit is always read as 0.<br>0 eTSEC continues waiting for the TxBD ring 0 poll timer to expire.<br>1 eTSEC immediately fetches a new TxBD from ring 0. |
| 30 | WWR | Write with response. This bit gives the user the assurance that a BD was updated in memory before it receives an interrupt concerning a transmit or receive frame.<br>0 Do not wait for acknowledgement from system for BD writes before setting IEVENT bits.<br>1 Before setting IEVENT bits TXB, TXF, TXE, XFUN, LC, CRL, RXB, RXF, the eTSEC waits for acknowledgement from system that the transmit or receive BD being updated was stored in memory. |
| 31 | WOP | Wait or poll for TxBD ring 0. This bit, which is applicable only to the transmitter and when TCTRL[TXSCHED] = 00, provides the user the option for the eTSEC to periodically poll TxBDs or to wait for software to tell eTSEC to fetch a buffer descriptor. While operating in the “Wait” mode, the eTSEC allows two additional reads of a descriptor which is not ready before entering a halt state. No interrupt is driven. To resume transmission, software must clear TSTAT[THLT].<br>0 Poll TxBD on ring 0 every 512 serial clocks.<br>1 Do not poll, but wait for TSTAT[THLT] to be cleared by the user. |

#### 14.5.3.1.9 TBI Physical Address Register (TBIPA)

The TBIPA, shown in Figure 14-10, is writable by the user to assign a physical address to the TBI (or RTBI) for MII management configuration. The TBI registers are accessed at the offset of TBIPA. For detailed descriptions of the TBI registers (the MII register set for the ten-bit interface) refer to Section 14.5.4, “Ten-Bit Interface (TBI).”

Offset eTSEC1:0x2_4030;
eTSEC3:0x2_6030

Access: Read/Write

| | 0–26 | 27–31 |
|---|---|---|
| R<br>W | — | TBIPA |
| Reset | All zeros | |

**Figure 14-10. TBIPA Register Definition**

Table 14-15 describes the fields of the TBIPA register.

**Table 14-15. TBIPA Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–26 | — | Reserved |
| 27–31 | TBIPA | This field is used to program the PHY address of the ten-bit interface’s MII management bus. To access the TBI register the user must write the TBIPA value to the MIIMADD [PHY Address] register located in the MAC register section. PHY Address 0 is reserved. Refer to Section 14.5.3.5.8, “MII Management Address Register (MIIMADD).” |

### 14.5.3.2 eTSEC Transmit Control and Status Registers

This section describes the control and status registers that are used specifically for transmitting Ethernet frames. All of the registers are 32 bits wide.

#### 14.5.3.2.1 Transmit Control Register (TCTRL)

This register is writable by the user to configure the transmit block. Figure 14-11 describes the TCTRL register.

Offset eTSEC1:0x2_4100;
eTSEC3:0x2_6100

Access: Mixed

| | 0–16 | 17 | 18 | 19 | 20 | 21–26 | 27 | 28 | 29–30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|
| R | — | IPCSEN | TUCSEN | VLINS | THDF | — | RFC_PAUSE | TFC_PAUSE | TXSCHED | — |
| W | | | | | | | | | | |
| Reset | All zeros | | | | | | | | | |

**Figure 14-11. TCTRL Register Definition**

Table 14-16 describes the fields of the TCTRL register.

**Table 14-16. TCTRL Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–16 | — | Reserved |
| 17 | IPCSEN | IP header checksum generation enable. When set, the eTSEC offloads IPv4 header checksum generation. See Section 14.6.4.2, “Transmit Path Off-Load and Tx PTP Packet Parsing,” on page 14-178.<br>0 IP header checksum generation is disabled even if enabled in a transmit frame control block.<br>1 IP header checksum generation is performed for IPv4 headers as determined by the settings in the current transmit frame control block. |
| 18 | TUCSEN | TCP/UDP header checksum generation enable. When set, the eTSEC offloads TCP or UDP header checksum generation. See Section 14.6.4.2, “Transmit Path Off-Load and Tx PTP Packet Parsing,” on page 14-178.<br>0 TCP or UDP header checksum generation is disabled even if enabled in a transmit frame control block.<br>1 TCP or UDP header checksum generation is performed as determined by the settings in the current transmit frame control block. |
| 19 | VLINS | VLAN (IEEE Std. 802.1Q) tag insertion enable. Applicable only for transmission through the Ethernet MAC.<br>0 Do not insert a VLAN tag into the frame.<br>1 Insert a VLAN tag into the frame. If the frame FCB has a valid VLAN field, use the FCB to source the VLAN control word, otherwise take the default VLAN control word from register DFVLAN. |
| 20 | THDF | Transmit half-duplex flow control under software control for 10-/100-Mbps half-duplex media. This bit is not self-resetting.<br>0 Disable back pressure<br>1 Back pressure is applied to media by raising carrier |
| 21–26 | — | Reserved |
| 27 | RFC_PAUSE | Receive flow control pause frame (written by the eTSEC). This read-only status bit is set if a flow control pause frame was received and the transmitter is paused for the duration defined in the received pause frame. This bit automatically clears after the pause duration is complete.<br>0 Pause duration complete.<br>1 Flow control pause frame received. |
| 28 | TFC_PAUSE | Transmit flow control pause frame. Set this bit to transmit a PAUSE frame. If this bit is set, the MAC stops transmission of data frames after the currently transmitting frame completes. Next, the MAC transmits a pause control frame with the duration value obtained from the PTV register. The TXC event occurs after sending the pause control frame. Finally, the controller clears TFC_PAUSE and resumes transmitting data frames as before. Note that pause control frames can still be transmitted if the Tx controller is stopped due to user assertion of DMACTRL[GTS] or reception of a PAUSE frame.<br>0 No request for Tx PAUSE frame pending or transmission complete.<br>1 Software request for Tx PAUSE frame pending. |

**Table 14-16. TCTRL Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 29–30 | TXSCHED | Transmit ring scheduling algorithm. This field determines which scheme the transmit scheduler uses to arbitrate between the enabled TxBD rings. The scheme chosen also controls how the DMACTRL and TQUEUE bits are interpreted. Ring polling is supported only by mode 00; the other modes require software to restart rings with the TSTAT register. TCP/IP offload can be enabled with any scheduling mode.<br>00 Single polled ring mode. TxBD ring 0 is the only ring serviced, even if other rings are enabled and ready. In this scheduler mode, the DMACTRL[WOP] and DMACTRL[TOD] bits control polling and retry behavior. This mode supports ring polling, and allows fetching of a non-ready TxBD to be retried twice.<br>01 Priority scheduling mode. Frames from enabled TxBD rings are serviced in ascending ring index order.<br>10 Modified weighted round-robin scheduling mode. Each TxBD ring is polled in sequence for frames that are ready for transmission. If a non-ready TxBD is fetched from a ring, that ring is removed from the scheduling pool until software re-enables it. Ready frames are repeatedly transmitted from a chosen ring until its transmission quota is exhausted. The transmission quota for TxBD ring *n* is set to WT*n* × 64 bytes, where WT*n* is a weight from the TR03WT/TR47WT registers. If a ring transmits more data than its quota allows, the excess is deducted from its quota on the next transmission opportunity, thereby preventing large frames from monopolizing the eTSEC bandwidth.<br>11 Reserved |
| 31 | — | Reserved |

### 14.5.3.2.2 Transmit Status Register (TSTAT)

This register is read/write-one-to-clear and is written by the eTSEC to convey DMA status information for each TxBD ring. The halt bit only has meaning for enabled rings. After processing transmit-related interrupts, software should use TSTAT to restart transmission from rings that may have been affected by the interrupt condition. In particular, an error condition that prevents eTSEC from continuing transmission halts DMA from all rings, including the ring that gave rise to the error. Figure 14-12 describes the TSTAT register.

Offset eTSEC1:0x2_4104;
eTSEC3:0x2_6104

Access: w1c

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|
| R | THLT0 | THLT1 | THLT2 | THLT3 | THLT4 | THLT5 | THLT6 | THLT7 | — | |
| W | w1c | w1c | w1c | w1c | w1c | w1c | w1c | w1c | | |
| Reset | All zeros | | | | | | | | | |

| | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|
| R | TXF0 | TXF1 | TXF2 | TXF3 | TXF4 | TXF5 | TXF6 | TXF7 | — | |
| W | w1c | w1c | w1c | w1c | w1c | w1c | w1c | w1c | | |
| Reset | All zeros | | | | | | | | | |

**Figure 14-12. TSTAT Register Definition**

Table 14-17 describes the fields of the TSTAT register.

**Table 14-17. TSTAT Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | THLT0 | Transmit halt of ring 0. Set by the eTSEC if is no longer processing transmit frames from this TxBD ring, and DMA from this ring is disabled. To re-start transmission from this TxBD ring, this bit must be cleared by writing 1 to it. This bit is set only on a general error condition (as in IEVENT[TXE]), regardless of TQUEUE[EN0], or if no ready TxBDs can be fetched. DMACTRL[GTS] being set by the user does not cause this bit to be set. Software should examine the halted queue's buffer descriptors for repeatable error conditions before taking it out of the halt state. Failure to do so may cause an effective livelock, in which the error condition recurs and halts all queues again.<br>Repeatable error conditions which cause halt include:<br>Bus error:<br>• Invalid BD or data address<br>• Uncorrectable error on BD or data read<br>TxBD programming errors:<br>• Ready=1 and length=0 |
| 1 | THLT1 | Transmit halt of ring 1. Set by the eTSEC if is no longer processing transmit frames from this TxBD ring, and DMA from this ring is disabled. To re-start transmission from this TxBD ring, this bit must be cleared by writing 1 to it. This bit is set only on a general error condition (as in IEVENT[TXE]), regardless of TQUEUE[EN1], or if no ready TxBDs can be fetched.DMACTRL[GTS] being set by the user does not cause this bit to be set.<br>Software should examine the halted queue's buffer descriptors for repeatable error conditions before taking it out of the halt state. Failure to do so may cause an effective livelock, in which the error condition recurs and halts all queues again.<br>Repeatable error conditions which cause halt include:<br>Bus error:<br>• Invalid BD or data address<br>• Uncorrectable error on BD or data read<br>TxBD programming errors:<br>• Ready=1 and length=0 |
| 2 | THLT2 | Transmit halt of ring 2. Set by the eTSEC if is no longer processing transmit frames from this TxBD ring, and DMA from this ring is disabled. To re-start transmission from this TxBD ring, this bit must be cleared by writing 1 to it. This bit is set only on a general error condition (as in IEVENT[TXE]), regardless of TQUEUE[EN2], or if no ready TxBDs can be fetched. DMACTRL[GTS] being set by the user does not cause this bit to be set.<br>Software should examine the halted queue's buffer descriptors for repeatable error conditions before taking it out of the halt state. Failure to do so may cause an effective livelock, in which the error condition recurs and halts all queues again.<br>Repeatable error conditions which cause halt include:<br>Bus error:<br>• Invalid BD or data address<br>• Uncorrectable error on BD or data read<br>TxBD programming errors:<br>• Ready=1 and length=0 |

**Table 14-17. TSTAT Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 3 | THLT3 | Transmit halt of ring 3. Set by the eTSEC if is no longer processing transmit frames from this TxBD ring, and DMA from this ring is disabled. To re-start transmission from this TxBD ring, this bit must be cleared by writing 1 to it. This bit is set only on a general error condition (as in IEVENT[TXE]), regardless of TQUEUE[EN3], or if no ready TxBDs can be fetched. DMACTRL[GTS] being set by the user does not cause this bit to be set.<br>Software should examine the halted queue's buffer descriptors for repeatable error conditions before taking it out of the halt state. Failure to do so may cause an effective livelock, in which the error condition recurs and halts all queues again.<br>Repeatable error conditions which cause halt include:<br>Bus error:<br>• Invalid BD or data address<br>• Uncorrectable error on BD or data read<br>TxBD programming errors:<br>• Ready=1 and length=0 |
| 4 | THLT4 | Transmit halt of ring 4. Set by the eTSEC if is no longer processing transmit frames from this TxBD ring, and DMA from this ring is disabled. To re-start transmission from this TxBD ring, this bit must be cleared by writing 1 to it. This bit is set only on a general error condition (as in IEVENT[TXE]), regardless of TQUEUE[EN4], or if no ready TxBDs can be fetched. DMACTRL[GTS] being set by the user does not cause this bit to be set.<br>Software should examine the halted queue's buffer descriptors for repeatable error conditions before taking it out of the halt state. Failure to do so may cause an effective livelock, in which the error condition recurs and halts all queues again.<br>Repeatable error conditions which cause halt include:<br>Bus error:<br>• Invalid BD or data address<br>• Uncorrectable error on BD or data read<br>TxBD programming errors:<br>• Ready=1 and length=0 |
| 5 | THLT5 | Transmit halt of ring 5. Set by the eTSEC if is no longer processing transmit frames from this TxBD ring, and DMA from this ring is disabled. To re-start transmission from this TxBD ring, this bit must be cleared by writing 1 to it. This bit is set only on a general error condition (as in IEVENT[TXE]), regardless of TQUEUE[EN5], or if no ready TxBDs can be fetched. DMACTRL[GTS] being set by the user does not cause this bit to be set.<br>Software should examine the halted queue's buffer descriptors for repeatable error conditions before taking it out of the halt state. Failure to do so may cause an effective livelock, in which the error condition recurs and halts all queues again.<br>Repeatable error conditions which cause halt include:<br>Bus error:<br>• Invalid BD or data address<br>• Uncorrectable error on BD or data read<br>TxBD programming errors:<br>• Ready=1 and length=0 |

Table 14-17. TSTAT Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 6 | THLT6 | Transmit halt of ring 6. Set by the eTSEC if is no longer processing transmit frames from this TxBD ring, and DMA from this ring is disabled. To re-start transmission from this TxBD ring, this bit must be cleared by writing 1 to it. This bit is set only on a general error condition (as in IEVENT[TXE]), regardless of TQUEUE[EN6], or if no ready TxBDs can be fetched. DMACTRL[GTS] being set by the user does not cause this bit to be set.<br>Software should examine the halted queue's buffer descriptors for repeatable error conditions before taking it out of the halt state. Failure to do so may cause an effective livelock, in which the error condition recurs and halts all queues again.<br>Repeatable error conditions which cause halt include:<br>Bus error:<br>• Invalid BD or data address<br>• Uncorrectable error on BD or data read<br>TxBD programming errors:<br>• Ready=1 and length=0 |
| 7 | THLT7 | Transmit halt of ring 7. Set by the eTSEC if is no longer processing transmit frames from this TxBD ring, and DMA from this ring is disabled. To re-start transmission from this TxBD ring, this bit must be cleared by writing 1 to it. This bit is set only on a general error condition (as in IEVENT[TXE]), regardless of TQUEUE[EN7], or if no ready TxBDs can be fetched. DMACTRL[GTS] being set by the user does not cause this bit to be set.<br>Software should examine the halted queue's buffer descriptors for repeatable error conditions before taking it out of the halt state. Failure to do so may cause an effective livelock, in which the error condition recurs and halts all queues again.<br>Repeatable error conditions which cause halt include:<br>Bus error:<br>• Invalid BD or data address<br>• Uncorrectable error on BD or data read<br>TxBD programming errors:<br>• Ready=1 and length=0 |
| 8–15 | — | Reserved |
| 16 | TXF0 | Transmit frame event occurred on ring 0. Set by the eTSEC if IEVENT[TXF] was set in relation to transmitting a frame from this ring. |
| 17 | TXF1 | Transmit frame event occurred on ring 1. Set by the eTSEC if IEVENT[TXF] was set in relation to transmitting a frame from this ring. |
| 18 | TXF2 | Transmit frame event occurred on ring 2. Set by the eTSEC if IEVENT[TXF] was set in relation to transmitting a frame from this ring. |
| 19 | TXF3 | Transmit frame event occurred on ring 3. Set by the eTSEC if IEVENT[TXF] was set in relation to transmitting a frame from this ring. |
| 20 | TXF4 | Transmit frame event occurred on ring 4. Set by the eTSEC if IEVENT[TXF] was set in relation to transmitting a frame from this ring. |
| 21 | TXF5 | Transmit frame event occurred on ring 5. Set by the eTSEC if IEVENT[TXF] was set in relation to transmitting a frame from this ring. |
| 22 | TXF6 | Transmit frame event occurred on ring 6. Set by the eTSEC if IEVENT[TXF] was set in relation to transmitting a frame from this ring. |

**Table 14-17. TSTAT Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 23 | TXF7 | Transmit frame event occurred on ring 7. Set by the eTSEC if IEVENT[TXF] was set in relation to transmitting a frame from this ring. |
| 24–31 | — | Reserved |

### 14.5.3.2.3 Default VLAN Control Word Register (DFVLAN)

This register defines the default value for the VLAN Ethertype and control word when VLAN tags are automatically inserted by the eTSEC, and no per-frame VLAN data is supplied by software. On receive, this register defines a customizable VLAN Ethertype for automatic deletion. Note that an Ethertype of 0x8808 (Control Word) is not permitted as a custom VLAN tag. Frames with an Ethertype of 0x8808 are dropped by the receiver. In the case of frames containing stacked VLAN tags, this register defines the tag associated with the outer or metropolitan area VLAN. Figure 14-13 describes the DFVLAN register.

Offset eTSEC1:0x2_4108;
eTSEC3:0x2_6108

Access: Read/Write

| | 0 – 15 | 16 – 18 | 19 | 20 – 31 |
|---|---|---|---|---|
| R<br>W | TAG | PRI | CFI | VID |
| Reset | 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 | 0 0 0 | 0 | 0 0 0 0 0 0 0 0 0 0 0 0 |

**Figure 14-13. DFVLAN Register Definition**

Table 14-18 describes the fields of the DFVLAN register.

**Table 14-18. DFVLAN Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–15 | TAG | This is the default Ethertype used to tag VLAN frames. On transmit, this tag is inserted ahead of the VLAN control word; TAG should be set to 0x8100 for IEEE 802.1Q VLAN. On receive, an Ethertype matching TAG or an Ethertype of 0x8100 marks a VLAN-tagged frame.<br>Note that if using DFVLAN to set a custom ethertype (that is, using a value other than 0x8100), packets received with a custom tag are not counted by any of the RMON counters. Affected counters include TRMGV, RMCA, RBCA, RXCF, RXPF, RXUO, RALN, RFLR, ROVR, RJBR, TMCA, TBCA, TXPF, TXCF. |
| 16–18 | PRI | This is the default value used for the IEEE Std. 802.1p frame priority. |
| 19 | CFI | This is the default value used for the IEEE Std. 802.1Q canonical format indicator. |
| 20–31 | VID | This is the default value used for the virtual-LAN identifier in VLAN-tagged frames. A value of zero is defined as the null VLAN, however field PRI may be still set independently. |

#### 14.5.3.2.4 Transmit Interrupt Coalescing Register (TXIC)

The TXIC register enables and configures the operational parameters for interrupt coalescing associated with transmitted frames. Figure 14-14 describes the definition for the TXIC register.

Offset eTSEC1:0x2_4110;
eTSEC3:0x2_6110

Access: Read/Write

| | 0 | 1 | 2 | 3 10 | 11 15 | 16 31 |
|---|---|---|---|---|---|---|
| R<br>W | ICEN | ICCS | — | ICFT | — | ICTT |
| Reset | All zeros | | | | | |

**Figure 14-14. TXIC Register Definition**

Table 14-19 describes the fields of the TXIC register.

**Table 14-19. TXIC Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | ICEN | Interrupt coalescing enable<br>0 Interrupt coalescing is disabled. Interrupts are raised as they are received.<br>1 Interrupt coalescing is enabled. If the eTSEC transmit frame interrupt is enabled (IMASK[TXFEN] is set), an interrupt is raised when the threshold number of frames is reached (defined by TXIC[ICFT]) or when the threshold timer expires (determined by TXIC[ICTT]). |
| 1 | ICCS | Interrupt coalescing timer clock source.<br>0 The coalescing timer advances count every 64 eTSEC Tx interface clocks (TSEC*n*_GTX_CLK).<br>1 The coalescing timer advances count every 64 system clocks[1]. This mode is recommended for FIFO operation. |
| 2 | — | Reserved |
| 3–10 | ICFT | Interrupt coalescing frame count threshold. While interrupt coalescing is enabled (TXIC[ICEN] is set), this value determines how many frames are transmitted before raising an interrupt. The eTSEC threshold counter is reset to ICFT following an interrupt. The value of ICFT must be greater than zero to avoid unpredictable behavior. |
| 11–15 | — | Reserved |
| 16–31 | ICTT | Interrupt coalescing timer threshold. While interrupt coalescing is enabled (TXIC[ICEN] is set), this value determines the maximum amount of time after transmitting a frame before raising an interrupt. If frames have been transmitted but the frame count threshold has not been met, an interrupt is raised when the threshold timer reaches zero. The threshold timer is reset to the value in this field and begins counting down upon transmission of the first frame having its TxBD[I] bit set. The threshold value is represented in units of 64 clock periods as specified by the timer clock source (TXIC[ICCS[). The value of ICTT must be greater than zero to avoid unpredictable behavior. |

[1] The term ‘system clock’ refers to CCB clock/2.

### 14.5.3.2.5 Transmit Queue Control Register (TQUEUE)

The TQUEUE register, shown in Figure 14-15, selectively enables each of the TxBD rings 0–7. By default, TxBD ring 0 is enabled.

Offset eTSEC1:0x2_4114;
eTSEC3:0x2_6114

Access: Read/Write

<table>
<tr><td></td><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td></td><td></td><td></td><td></td><td></td><td></td><td>31</td></tr>
<tr><td>R<br>W</td><td colspan="16">—</td><td>EN0</td><td>EN1</td><td>EN2</td><td>EN3</td><td>EN4</td><td>EN5</td><td>EN6</td><td>EN7</td><td colspan="8">—</td></tr>
<tr><td>Reset</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>

**Figure 14-15. TQUEUE Register Definition**

Table 14-20 describes the TQUEUE register.

**Table 14-20. TQUEUE Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–15 | — | Reserved |
| 16 | EN0 | Transmit queue 0 enable.<br>0 TxBD ring is not queried for transmission. In effect the transmit queue is disabled.<br>1 TxBD ring is queried for transmission. |
| 17 | EN1 | Transmit queue 1 enable.<br>0 TxBD ring is not queried for transmission. In effect the transmit queue is disabled.<br>1 TxBD ring is queried for transmission. |
| 18 | EN2 | Transmit queue 2 enable.<br>0 TxBD ring is not queried for transmission. In effect the transmit queue is disabled.<br>1 TxBD ring is queried for transmission. |
| 19 | EN3 | Transmit queue 3 enable.<br>0 TxBD ring is not queried for transmission. In effect the transmit queue is disabled.<br>1 TxBD ring is queried for transmission. |
| 20 | EN4 | Transmit queue 4 enable.<br>0 TxBD ring is not queried for transmission. In effect the transmit queue is disabled.<br>1 TxBD ring is queried for transmission. |
| 21 | EN5 | Transmit queue 5 enable.<br>0 TxBD ring is not queried for transmission. In effect the transmit queue is disabled.<br>1 TxBD ring is queried for transmission. |
| 22 | EN6 | Transmit queue 6 enable.<br>0 TxBD ring is not queried for transmission. In effect the transmit queue is disabled.<br>1 TxBD ring is queried for transmission. |
| 23 | EN7 | Transmit queue 7 enable.<br>0 TxBD ring is not queried for transmission. In effect the transmit queue is disabled.<br>1 TxBD ring is queried for transmission. |
| 24–31 | — | Reserved |

#### 14.5.3.2.6 TxBD Ring 0–3 Weighting Register (TR03WT)

When modified weighted round-robin Tx scheduling is enabled (TCTRL[TXSCHED] = 10), this register determines the weighting applied to each transmit queue for queues 0 to 3. For priority-based scheduling, TR03WT has no effect. A description of how queue weights affect eTSEC’s round-robin algorithm appears in Section 14.6.5.3.2, “Modified Weighted Round-Robin Queuing (MWRR).” Figure 14-16 describes the TR03WT register.

Offset eTSEC1:0x2_4140;
eTSEC3:0x2_6140

Access: Read/Write

| | 0–7 | 8–15 | 16–23 | 24–31 |
|---|---|---|---|---|
| R<br>W | WT0 | WT1 | WT2 | WT3 |
| Reset | All zeros | | | |

**Figure 14-16. TR03WT Register Definition**

Table 14-21 describes the fields of the TR03WT register.

**Table 14-21. TR03WT Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–7 | WT0 | Weighting value for TxBD ring 0 when TCTRL[TXSCHED] = 10. On each round of the Tx scheduler, a minimum of WT0 × 64 bytes of data are scheduled for transmission from TxBD ring 0. Clearing this field prevents transmission. |
| 8–15 | WT1 | Weighting value for TxBD ring 1 when TCTRL[TXSCHED] = 10. On each round of the Tx scheduler, a minimum of WT1 × 64 bytes of data are scheduled for transmission from TxBD ring 1. Clearing this field prevents transmission. |
| 16–23 | WT2 | Weighting value for TxBD ring 2 when TCTRL[TXSCHED] = 10. On each round of the Tx scheduler, a minimum of WT2 × 64 bytes of data are scheduled for transmission from TxBD ring 2. Clearing this field prevents transmission. |
| 24–31 | WT3 | Weighting value for TxBD ring 3 when TCTRL[TXSCHED] = 10. On each round of the Tx scheduler, a minimum of WT3 × 64 bytes of data are scheduled for transmission from TxBD ring 3. Clearing this field prevents transmission. |

#### 14.5.3.2.7 TxBD Ring 4–7 Weighting Register (TR47WT)

When modified weighted round-robin Tx scheduling is enabled (TCTRL[TXSCHED] = 10), this register determines the weighting applied to each enabled transmit queue for queues 4 to 7. For priority-based scheduling, TR47WT has no effect. A description of how queue weights affect eTSEC’s modified weighted round-robin algorithm appears in Section 14.6.5.3.2, “Modified Weighted Round-Robin Queuing (MWRR).” Figure 14-17 describes the definition for the TR47WT register.

Offset eTSEC1:0x2_4144;
eTSEC3:0x2_6144

Access: Read/Write

| | 0–7 | 8–15 | 16–23 | 24–31 |
|---|---|---|---|---|
| R<br>W | WT4 | WT5 | WT6 | WT7 |
| Reset | All zeros | | | |

**Figure 14-17. TR47WT Register Definition**

Table 14-22 describes the fields of the TR47WT register.

Table 14-22. TR47WT Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–7 | WT4 | Weighting value for TxBD ring 4 when TCTRL[TXSCHED] = 10. On each round of the Tx scheduler, a minimum of WT4 × 64 bytes of data are scheduled for transmission from TxBD ring 4. Clearing this field prevents transmission. |
| 8–15 | WT5 | Weighting value for TxBD ring 5 when TCTRL[TXSCHED] = 10. On each round of the Tx scheduler, a minimum of WT5 × 64 bytes of data are scheduled for transmission from TxBD ring 5. Clearing this field prevents transmission. |
| 16–23 | WT6 | Weighting value for TxBD ring 6 when TCTRL[TXSCHED] = 10. On each round of the Tx scheduler, a minimum of WT6 × 64 bytes of data are scheduled for transmission from TxBD ring 6. Clearing this field prevents transmission. |
| 24–31 | WT7 | Weighting value for TxBD ring 7 when TCTRL[TXSCHED] = 10. On each round of the Tx scheduler, a minimum of WT7 × 64 bytes of data are scheduled for transmission from TxBD ring 7. Clearing this field prevents transmission. |

### 14.5.3.2.8 Transmit Data Buffer Pointer High Register (TBDBPH)

The TBDBPH register is written by the user with the most significant address bits common to all TxBD buffer addresses, TxBD[Data Buffer Pointer]. As a consequence, all Tx buffers must be placed in a 4 gigabyte segment of memory whose base address is prefixed by the bits in TBDBPH. The TxBD ring itself can reside in a different memory region (based at TBASEH). Figure 14-18 describes the definition for the TBDBPH register.

Offset eTSEC1:0x2_4180;
eTSEC3:0x2_6180

Access: Read/Write

| | 0–27 | 28–31 |
|---|---|---|
| R<br>W | — | TBDBPH |
| Reset | All zeros | |

Figure 14-18. TBDBPH Register Definition

Table 14-25 describes the fields of the TBDBPH register.

Table 14-23. TBDBPH Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–27 | — | Reserved |
| 28–31 | TBDBPH | Most significant bits common to all data buffer addresses contained in TxBDs. The user must initialize TBDBPH before enabling the eTSEC transmit function. |

### 14.5.3.2.9 Transmit Buffer Descriptor Pointers 0–7 (TBPTR0–TBPTR7)

TBPTR0–TBPTR7 each contains the low-order 32 bits of the next transmit buffer descriptor address for their respective TxBD ring. Figure 14-19 describes the TBPTR registers. These registers takes on the value of their ring’s associated TBASE when the TBASE register is written by software. Software must not write TBPTR0–TBPTR7 while eTSEC is actively transmitting frames. However, TBPTR0– TBPTR7 can be modified when the transmitter is disabled or when no Tx buffer is in use (after a GRACEFUL STOP TRANSMIT command is issued and the frame completes its transmission) in order to change the next TxBD eTSEC transmits.

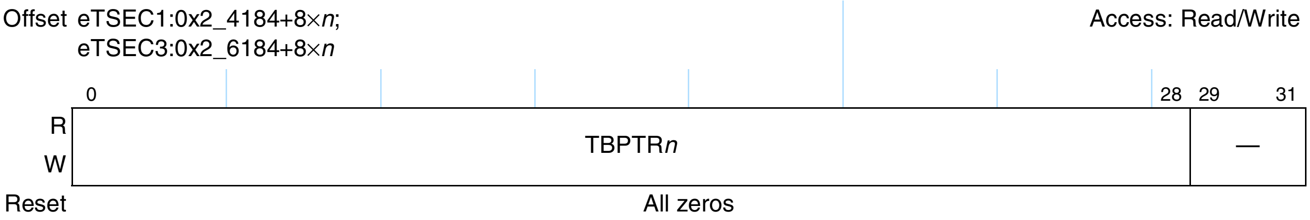


**Figure 14-19. TBPTR0–TBPTR7 Register Definition**

Table 14-24 describes the fields of the TBPTR*n* register.

**Table 14-24. TBPTR*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–28 | TBPTR*n* | Current TxBD pointer for TxBD ring *n*. Points to the current BD being processed or to the next BD the transmitter uses when it is idling. When the end of the TxBD ring is reached, eTSEC initializes TBPTR*n* to the value in the corresponding TBASE*n*. The TBPTR register is internally written by the eTSEC's DMA controller during transmission. The pointer increments by eight (bytes) each time a descriptor is closed successfully by the eTSEC. Note that the three least significant bits of this register are read-only and zero. After an error condition, the eTSEC returns TBPTR*n* to point to the first BD of the frame partially transmitted. |
| 29–31 | — | Reserved |

### 14.5.3.2.10 Transmit Descriptor Base Address High Register (TBASEH)

The TBASEH register is written by the user with the most significant address bits common to all TxBD addresses, including TBASE0–TBASE7 and TBPTR0–TBPTR7. As a consequence, all TxBD rings must be placed in a 4 Gbyte segment of memory whose base address is prefixed by the bits in TBASEH. Data buffers are located in a potentially different region, based at TBDBPH. Figure 14-20 describes the TBASEH register.

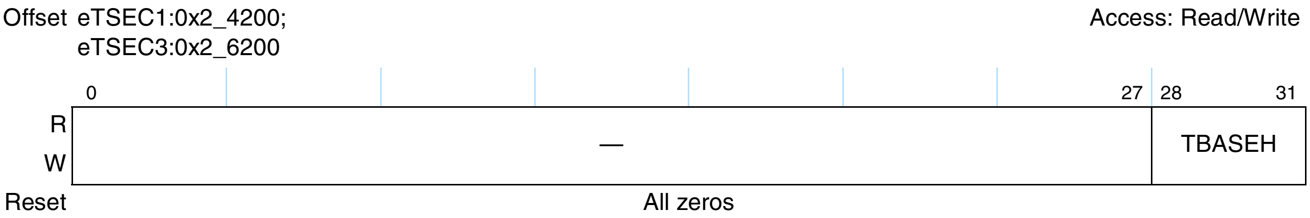


**Figure 14-20. TBASEH Register Definition**

Table 14-25 describes the fields of the TBASEH register.

**Table 14-25. TBASEH Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–27 | — | Reserved |
| 28–31 | TBASEH | Most significant bits common to all TxBD addresses—except data buffer pointers. The user must initialize TBASEH before enabling the eTSEC transmit function. |

### 14.5.3.2.11 Transmit Descriptor Base Address Registers (TBASE0–TBASE7)

The TBASE*n* registers are written by the user with the base address of each TxBD ring *n*. Each such value must be divisible by eight, since the three least significant bits always write as 000. Figure 14-21 describes the definition for the TBASE*n* registers.

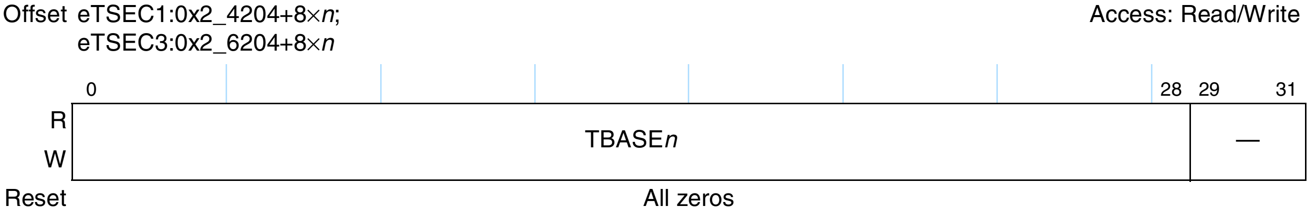


**Figure 14-21. TBASE Register Definition**

Table 14-26 describes the fields of the TBASE*n* registers.

**Table 14-26. TBASE0–TBASE7 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–28 | TBASE*n* | Transmit base for ring *n*. TBASE defines the starting location in the memory map for the eTSEC TxBDs. This field must be 8-byte aligned. Together with setting the W (wrap) bit in the last BD, the user can select how many BDs to allocate for the transmit packets. The user must initialize TBASE before enabling the eTSEC transmit function on the associated ring. |
| 29–31 | — | Reserved |

### 14.5.3.2.12 Transmit Time Stamp Identification Register (TMR_TXTS1–2_ID)

Transmit time stamp identification register (TMR_TXTS*n*_ID). This register holds the identification number of the transmitted frame corresponding to the timestamp captured in TMR_TXTS*n*_H/L. Each time the eTSEC is instructed to capture the timestamp of an outgoing frame via TxFCB[PTP] the associated field in TxFCB[PTP_ID] is stored in this register, overwriting the previous value.

This register is read only in normal operation. Figure 14-22 describes the definition for the TMR_TXTS*n*_ID register.

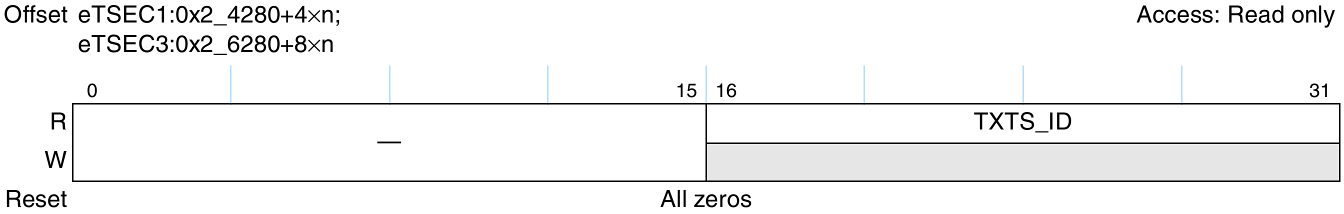


**Figure 14-22. TMR_TXTS*n*_ID Register Definition**

Table 14-27 describes the fields of the TMR_TXTS*n*_ID register.

**Table 14-27. TMR_TXTS*n*_ID Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–15 | — | Reserved |
| 16–31 | TXTS_ID | Tx time stamp identification field |

### 14.5.3.2.13 Transmit Time Stamp Register (TMR_TXTS1–2_H/L)

Transmit stamp register (TMR_TXTS*n*_H/L). This register holds the value of the TMR_CNT_H/L when a frame tagged for timestamp capture (via Tx FCB[PTP]) is transmitted. Upon transmission of the start of frame symbol of such a frame, the value in TMR_CNT_H/L is copied into TMR_TXTS*n*_H/L.

This register is read only in normal operation. Figure 14-23 depicts TMR_TXTS*n*_H/L.

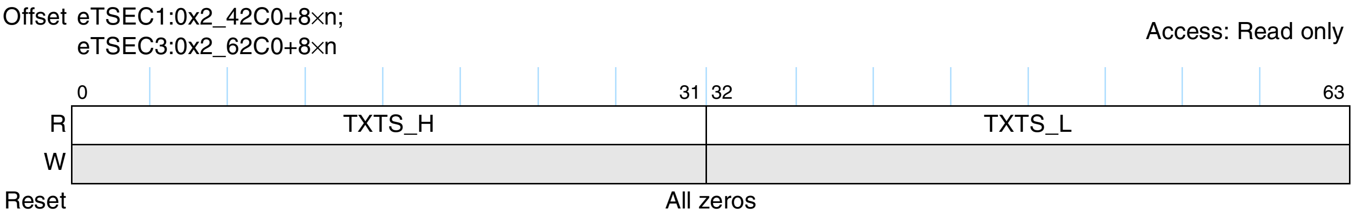


**Figure 14-23. TMR_TXTS*n*_H/L Register Definition**

Table 14-28 describes the fields of the TMR_TXTS*n*_H/L register.

**Table 14-28. TMR_TXTS*n*_H/L Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–63 | TXTS_H/L | Time stamp field of the transmitted PTP packet's start of frame detection. |

## 14.5.3.3 eTSEC Receive Control and Status Registers

This section describes the control and status registers that are used specifically for receiving Ethernet frames. All of the registers are 32 bits wide.

### 14.5.3.3.1 Receive Control Register (RCTRL)

The RCTRL register is programmed by the user and controls the operational mode of the receiver. It must be written only after a system reset (at initialization) or after a graceful receive stop has completed. Figure 14-24 describes the RCTRL register.

Offset eTSEC1:0x2_4300;
eTSEC3:0x2_6300

Access: Read/Write

| | 0–6 | 7 | 8–10 | 11–15 |
|---|---|---|---|---|
| R<br>W | L2OFF | TS | — | PAL |
| Reset | All zeros | | | |

| | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24–25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | LFC | VLEX | FILREN | FSQEN | GHTX | IPCSEN | TUCSEN | PRSDEP | PRSFM | BC_REJ | PROM | RSF | EMEN | — |
| Reset | All zeros | | | | | | | | | | | | | | |

**Figure 14-24. RCTRL Register Definition**

Table 14-29 describes the fields of the RCTRL register.

**Table 14-29. RCTRL Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–6 | L2OFF | Layer 2 offset. The number of octet pairs from the start of the frame that the parser should expect to see before the first byte of the Ethernet DA.<br>For frames received over Ethernet, the start of frame is regarded as the SFD symbol.<br>For packets received through the FIFO packet interface the start of frame is regarded as the first octet of receive data.<br>The user may think of this value as representing the length - in multiples of two bytes - of a ‘shim’ header that is inserted between the SFD and DA. By writing to RCTRL with a mask of 0xFE00_0000, the even byte length restriction is guaranteed.<br>For normal frames, this field should be left as 0. |
| 7 | TS | Time stamp incoming packets as padding bytes. PAL field is set to 8 if the PAL field is programmed to less than 8. Must be set to zero if TMR_CTRL[TE]=0. |
| 8–10 | — | Reserved |
| 11–15 | PAL | Packet alignment padding length. If not zero, PAL (1–31) bytes of zero padding are inserted before the start of each received frame, but following the RxFCB if TOE is enabled. For Ethernet where optional preamble extraction is enabled, the padding appears before the preamble, otherwise the padding precedes the layer 2 header. The value of PAL can be set so that the start of the IP header in the receive data buffer is aligned to a 32-bit boundary. Normally, setting PAL = 2 provides minimal padding to ensure such alignment of the IP header.<br>Note that the minimum zero padding value for this field should be PAL–8 if the TS field is set and 0 when PAL is < 8. |

### Table 14-29. RCTRL Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 17 | LFC | Lossless flow control. When set, the eTSEC determines the number of free BDs (through RQPARM*n*[LEN] and RBTPTR*n*) in each active ring. Should the free BD count in an active ring drop below its setting for RQPARM*n*[FBTHR], the eTSEC asserts link layer flow control.<br>For full-duplex ethernet connections, the eTSEC emits a pause frame as if TCTRL[TFC_PAUSE] was set. For FIFO packet interface connections, the RFC signal is asserted.<br>0 Disabled. This is the default<br>1 Enabled, calculate the free BDs in each active ring and assert link layer flow control if required. |
| 18 | VLEX | Enable automatic VLAN tag extraction and deletion from Ethernet frames. Note that VLEX must be cleared if L2OFF is non-zero.<br>0 Do not delete VLAN tags from received Ethernet frames.<br>1 If a VLAN tag is seen after the Ethernet source address, and PRSDEP is non-zero, delete the VLAN tag and return the VLAN control word in the frame control block returned with this frame.<br>Note that if PRSDEP is cleared, VLEX must be cleared as well. (VLAN tag extraction is only supported when the parser is enabled.) |
| 19 | FILREN | Filer enable. When set, the receive frame filer is enabled. This file accepted frames to a particular RxBD ring according to rules defined in the filer table. In this case, PRSDEP must not be cleared.<br>0 Do not search the receive queue filer table for received frames. All received frames are sent to RxBD ring 0 by default.<br>1 Search the receive queue filer table for received frames, and let the filer determine the index of the RxBD ring for each frame.<br>Note that if PRSDEP is cleared, FILREN must be cleared as well. |
| 20 | FSQEN | Enable single-queue mode for the receive frame filer. This bit is ignored unless FILREN is also set.<br>0 The filer chooses the RxBD ring using the least significant bits of the virtual queue ID as a ring index.<br>1 The filer always attempts to file received frames to ring 0, regardless of virtual queue ID. This mode is intended for operating the filer as a packet classification engine. |
| 21 | GHTX | Group address hash table extend. By default, the group address hash table is 256 entries (as defined by registers GADDR0–GADDR7); registers IGADDR0–IGADDR7 are then used to define the individual address hash table. When this bit is set, the hash table is extended to a total of 512 entries (IGADDR0–IGADDR7 are then the first 256 entries of the extended 512-entry group address hash table).<br>0 Both the individual and group hash functions are the 8 MSBs of the CRC-32 of the Ethernet destination address.<br>1 The group hash function is the 9 MSBs of the CRC-32 of the Ethernet destination address. The individual address hash function is unavailable. |
| 22 | IPCSEN | IP Checksum verification enable. See Section 14.6.4.3, “Receive Path Off-Load.”<br>0 IPv4 header checksums are not verified by the eTSEC—even if layer 3 parsing is enabled.<br>1 Perform IPv4 header checksum verification if PRSDEP > 01. |
| 23 | TUCSEN | TCP or UDP Checksum verification enable. See Section 14.6.4.3, “Receive Path Off-Load.”<br>0 TCP or UDP checksums are not verified by the eTSEC—even if layer 4 parsing is enabled.<br>1 Perform TCP or UDP checksum verification if PRSDEP = 11. |

Table 14-29. RCTRL Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 24–25 | PRSDEP | Parser control. The level of parser layer recognition is determined as follows:<br>00 Parser disabled. Receive frame filer must also be disabled by clearing RCTRL[FILREN]. This should be the setting for raw (non-IP) packets received over a FIFO interface.<br>01 Only L2 (Ethernet) protocols are recognized. For packets received over a FIFO interface, this parse level is available only if RCTRL[PRSFM]=1.<br>10 L2 and L3 (IP) protocols are recognized over any interface not configured as a FIFO interface. If RCTRL[PRSFM]=0, this encoding means L3 (IP) only protocols are recognized for packets received over a FIFO interface. If RCTRL[PRSFM]=1, this encoding means L2 and L3 (IP) protocols are recognised for packets received over a FIFO interface.<br>11 L2, L3, and L4 (TCP/UDP) protocols are recognized over any interface not configured as a FIFO interface. If RCTRL[PRSFM]=0, this encoding means L3 and L4 (TCP/UDP) protocols are recognized for packets received over a FIFO interface. If RCTRL[PRSDEP]=1, this encoding means L2, L3, and L4 (TCP/UDP) protocols are recognized for packets received over a FIFO interface.<br>If this field is non-zero, a TOE frame control block is prepended to the received frame, and the first RxBD points to the FCB.<br>Note that if PRSDEP is cleared, VLEX must be cleared as well. (VLAN tag extraction is only supported when the parser is enabled.) Also, if PRSDEP is cleared, FILREN must also be cleared. |
| 26 | PRSFM | FIFO-mode parsing<br>0 L2 parsing in FIFO mode is not available. Must be 0 for non-FIFO modes.<br>1 L2 parsing in FIFO mode is available |
| 27 | BC_REJ | Broadcast frame reject. If this bit is set, frames with DA (destination address) = FFFF_FFFF_FFFF are rejected unless RCTRL[PROM] is set. If both BC_REJ and RCTRL[PROM] are set, then frames with broadcast DA are accepted and the M (MISS) bit is set in the receive BD. |
| 28 | PROM | Promiscuous mode. All Ethernet frames, regardless of destination address, are accepted. |
| 29 | RSF | Receive short frame mode. When set, enables the reception of frames shorter than 64 bytes. For packets received over the FIFO packet interface, this bit has no effect (packets shorter than 64 bytes are always accepted).<br>0 Ethernet frames less than 64B in length are silently dropped.<br>1) Frames more than 16B and less than 64B in length are accepted upon a DA match.<br>Note that frames less than or equal to 16B in length are always silently dropped. |
| 30 | EMEN | Exact match MAC address enable. If this bit is set, the MAC01ADDR1–MAC15ADDR1 and MAC01ADDR2–MAC15ADDR2 registers are recognized as containing MAC addresses aliasing the MAC's station address. Setting this bit therefore allows eTSEC to receive Ethernet frames having a destination address matching one of these 15 addresses. |
| 31 | — | Reserved |

### 14.5.3.3.2 Receive Status Register (RSTAT)

The eTSEC writes to this register under the following conditions:

- A frame interrupt event occurred on one or more RxBD rings
- The receiver runs out of descriptors due to a busy condition on a RxBD ring
- The receiver was halted because an error condition was encountered while receiving a frame

Writing 1 to any bit of this register clears it. Software should clear the QHLT bit to take eTSEC's receiver function out of halt state for the associated queue. Figure 14-25 describes the definition for the RSTAT register.

Offset eTSEC1:0x2_4304;
eTSEC3:0x2_6304

Access: w1c

| | 0–7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|
| R | — | QHLT0 | QHLT1 | QHLT2 | QHLT3 | QHLT4 | QHLT5 | QHLT6 | QHLT7 |
| W | | w1c | w1c | w1c | w1c | w1c | w1c | w1c | w1c |
| Reset | All zeros | | | | | | | | |

| | 16–23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|
| R | — | RXF0 | RXF1 | RXF2 | RXF3 | RXF4 | RXF5 | RXF6 | RXF7 |
| W | | w1c | w1c | w1c | w1c | w1c | w1c | w1c | w1c |
| Reset | All zeros | | | | | | | | |

**Figure 14-25. RSTAT Register Definition**

Table 14-30 describes the fields of the RSTAT register.

**Table 14-30. RSTAT Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–7 | — | Reserved |
| 8 | QHLT0 | RxBD queue 0 is halted. It is a hardware-initiated stop indication. (DMACTRL[GRS] being set by the user does not cause a QHLT0 to be set.). The current frame and all other frames directed to a halted queue are discarded. A write with a value of 1 re-enables the queue for receiving.<br>0 This queue is enabled for reception. (That is, it is not halted)<br>1 All controller receive activity to this queue is halted. |
| 9 | QHLT1 | RxBD queue 1 is halted. It is a hardware-initiated stop indication. (DMACTRL[GRS] being set by the user does not cause a QHLT1 to be set.). The current frame and all other frames directed to a halted queue are discarded. A write with a value of 1 re-enables the queue for receiving.<br>0 This queue is enabled for reception. (That is, it is not halted)<br>1 All controller receive activity to this queue is halted. |
| 10 | QHLT2 | RxBD queue 2 is halted. It is a hardware-initiated stop indication. (DMACTRL[GRS] being set by the user does not cause a QHLT2 to be set.). The current frame and all other frames directed to a halted queue are discarded. A write with a value of 1 re-enables the queue for receiving.<br>0 This queue is enabled for reception. (That is, it is not halted)<br>1 All controller receive activity to this queue is halted. |
| 11 | QHLT3 | RxBD queue 3 is halted. It is a hardware-initiated stop indication. (DMACTRL[GRS] being set by the user does not cause a QHLT3 to be set.). The current frame and all other frames directed to a halted queue are discarded. A write with a value of 1 re-enables the queue for receiving.<br>0 This queue is enabled for reception. (That is, it is not halted)<br>1 All controller receive activity to this queue is halted. |
| 12 | QHLT4 | RxBD queue 4 is halted. It is a hardware-initiated stop indication. (DMACTRL[GRS] being set by the user does not cause a QHLT4 to be set.). The current frame and all other frames directed to a halted queue are discarded. A write with a value of 1 re-enables the queue for receiving.<br>0 This queue is enabled for reception. (That is, it is not halted)<br>1 All controller receive activity to this queue is halted. |

**Table 14-30. RSTAT Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 13 | QHLT5 | RxBD queue 5 is halted. It is a hardware-initiated stop indication. (DMACTRL[GRS] being set by the user does not cause a QHLT5 to be set.). The current frame and all other frames directed to a halted queue are discarded. A write with a value of 1 re-enables the queue for receiving.<br>0 This queue is enabled for reception. (That is, it is not halted)<br>1 All controller receive activity to this queue is halted. |
| 14 | QHLT6 | RxBD queue 6 is halted. It is a hardware-initiated stop indication. (DMACTRL[GRS] being set by the user does not cause a QHLT6 to be set.). The current frame and all other frames directed to a halted queue are discarded. A write with a value of 1 re-enables the queue for receiving.<br>0 This queue is enabled for reception. (That is, it is not halted)<br>1 All controller receive activity to this queue is halted. |
| 15 | QHLT7 | RxBD queue 7 is halted. It is a hardware-initiated stop indication. (DMACTRL[GRS] being set by the user does not cause a QHLT7 to be set.). The current frame and all other frames directed to a halted queue are discarded. A write with a value of 1 re-enables the queue for receiving.<br>0 This queue is enabled for reception. (That is, it is not halted)<br>1 All controller receive activity to this queue is halted. |
| 16–23 | — | Reserved |
| 24 | RXF0 | Receive frame event occurred on ring 0. Set by the eTSEC if IEVENT[RXF] was set in relation to receiving a frame to this ring. |
| 25 | RXF1 | Receive frame event occurred on ring 1. Set by the eTSEC if IEVENT[RXF] was set in relation to receiving a frame to this ring. |
| 26 | RXF2 | Receive frame event occurred on ring 2. Set by the eTSEC if IEVENT[RXF] was set in relation to receiving a frame to this ring. |
| 27 | RXF3 | Receive frame event occurred on ring 3. Set by the eTSEC if IEVENT[RXF] was set in relation to receiving a frame to this ring. |
| 28 | RXF4 | Receive frame event occurred on ring 4. Set by the eTSEC if IEVENT[RXF] was set in relation to receiving a frame to this ring. |
| 29 | RXF5 | Receive frame event occurred on ring 5. Set by the eTSEC if IEVENT[RXF] was set in relation to receiving a frame to this ring. |
| 30 | RXF6 | Receive frame event occurred on ring 6. Set by the eTSEC if IEVENT[RXF] was set in relation to receiving a frame to this ring. |
| 31 | RXF7 | Receive frame event occurred on ring 7. Set by the eTSEC if IEVENT[RXF] was set in relation to receiving a frame to this ring. |

#### 14.5.3.3.3 Receive Interrupt Coalescing Register (RXIC)

The RXIC register enables and configures the operational parameters for interrupt coalescing associated with received frames. Figure 14-26 describes the RXIC register.

Offset eTSEC1:0x2_4310;<br>
eTSEC3:0x2_6310

Access: Read/Write

| | 0 | 1 | 2 | 3–10 | 11–15 | 16–31 |
|---|---|---|---|---|---|---|
| R<br>W | ICEN | ICCS | — | ICFT | — | ICTT |
| Reset | All zeros | | | | | |

**Figure 14-26. RXIC Register Definition**

Table 14-31 describes the fields of the RXIC register.

**Table 14-31. RXIC Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | ICEN | Interrupt coalescing enable<br>0 Interrupt coalescing is disabled. Interrupts are raised as they are received.<br>1 Interrupt coalescing is enabled. If the eTSEC receive frame interrupt is enabled (IMASK[RXFEN] is set), an interrupt is raised when the threshold number of frames is reached (defined by RXIC[ICFT]) or when the threshold timer expires (determined by RXIC[ICTT]). |
| 1 | ICCS | Interrupt coalescing timer clock source.<br>0 The coalescing timer advances count every 64 eTSEC Rx interface clocks (TSECn_GTX_CLK).<br>1 The coalescing timer advances count every 64 system clocks[1]. This mode is recommended for FIFO operation. |
| 2 | — | Reserved |
| 3–10 | ICFT | Interrupt coalescing frame count threshold. While interrupt coalescing is enabled (RXIC[ICE] is set), this value determines how many frames are received before raising an interrupt. The eTSEC threshold counter is reset to ICFT following an interrupt. The value of ICFT must be greater than zero avoid unpredictable behavior. |
| 11–15 | — | Reserved |
| 16–31 | ICTT | Interrupt coalescing timer threshold. While interrupt coalescing is enabled (RXIC[ICE] is set), this value determines the maximum amount of time after receiving a frame before raising an interrupt. If frames have been received but the frame count threshold has not been met, an interrupt is raised when the threshold timer reaches zero. The threshold timer is reset to the value in this field and begins counting down upon receiving the first frame having its RxBD[I] bit set. The threshold value is represented in units equal to 64 periods of the clock specified by RXIC[ICCS]. ICTT must be greater than zero to avoid unpredictable behavior. |

[1] The term ‘system clock’ refers to CCB clock/2.

### 14.5.3.3.4 Receive Queue Control Register (RQUEUE)

The RQUEUE register enables each of the RxBD rings 0–7. By default, RxBD ring 0 is enabled. Figure 14-27 describes the definition for the RQUEUE register.

Offset eTSEC1:0x2_4314;
eTSEC3:0x2_6314

Access: Read/Write

| | 0 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | EX0 | EX1 | EX2 | EX3 | EX4 | EX5 | EX6 | EX7 | — | EN0 | EN1 | EN2 | EN3 | EN4 | EN5 | EN6 | EN7 |
| Reset | 0 0 0 0 0 0 0 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 0 0 0 0 0 0 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Figure 14-27. RQUEUE Register Definition**

Table 14-32 describes the RQUEUE register.

**Table 14-32. RQUEUE Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–7 | — | Reserved |
| 8 | EX0 | Receive queue 0 extract enable.<br>0 Data transferred by DMA to this RxBD ring is not extracted to cache.<br>1 Data transferred by DMA to this RxBD ring undergoes extraction according to ATTR register. |
| 9 | EX1 | Receive queue 1 extract enable.<br>0 Data transferred by DMA to this RxBD ring is not extracted to cache.<br>1 Data transferred by DMA to this RxBD ring undergoes extraction according to ATTR register. |
| 10 | EX2 | Receive queue 2 extract enable.<br>0 Data transferred by DMA to this RxBD ring is not extracted to cache.<br>1 Data transferred by DMA to this RxBD ring undergoes extraction according to ATTR register. |
| 11 | EX3 | Receive queue 3 extract enable.<br>0 Data transferred by DMA to this RxBD ring is not extracted to cache.<br>1 Data transferred by DMA to this RxBD ring undergoes extraction according to ATTR register. |
| 12 | EX4 | Receive queue 4 extract enable.<br>0 Data transferred by DMA to this RxBD ring is not extracted to cache.<br>1 Data transferred by DMA to this RxBD ring undergoes extraction according to ATTR register. |
| 13 | EX5 | Receive queue 5 extract enable.<br>0 Data transferred by DMA to this RxBD ring is not extracted to cache.<br>1 Data transferred by DMA to this RxBD ring undergoes extraction according to ATTR register. |
| 14 | EX6 | Receive queue 6 extract enable.<br>0 Data transferred by DMA to this RxBD ring is not extracted to cache.<br>1 Data transferred by DMA to this RxBD ring undergoes extraction according to ATTR register. |
| 15 | EX7 | Receive queue 7 extract enable.<br>0 Data transferred by DMA to this RxBD ring is not extracted to cache.<br>1 Data transferred by DMA to this RxBD ring undergoes extraction according to ATTR register. |
| 16–23 | — | Reserved |
| 24 | EN0 | Receive queue 0 enable.<br>0 RxBD ring is not queried for reception. In effect the receive queue is disabled.<br>1 RxBD ring is queried for reception. |

**Table 14-32. RQUEUE Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 25 | EN1 | Receive queue 1 enable.<br>0 RxBD ring is not queried for reception. In effect the receive queue is disabled.<br>1 RxBD ring is queried for reception. |
| 26 | EN2 | Receive queue 2 enable.<br>0 RxBD ring is not queried for reception. In effect the receive queue is disabled.<br>1 RxBD ring is queried for reception. |
| 27 | EN3 | Receive queue 3 enable.<br>0 RxBD ring is not queried for reception. In effect the receive queue is disabled.<br>1 RxBD ring is queried for reception. |
| 28 | EN4 | Receive queue 4 enable.<br>0 RxBD ring is not queried for reception. In effect the receive queue is disabled.<br>1 RxBD ring is queried for reception. |
| 29 | EN5 | Receive queue 5 enable.<br>0 RxBD ring is not queried for reception. In effect the receive queue is disabled.<br>1 RxBD ring is queried for reception. |
| 30 | EN6 | Receive queue 6 enable.<br>0 RxBD ring is not queried for reception. In effect the receive queue is disabled.<br>1 RxBD ring is queried for reception. |
| 31 | EN7 | Receive queue 7 enable.<br>0 RxBD ring is not queried for reception. In effect the receive queue is disabled.<br>1 RxBD ring is queried for reception. |

#### 14.5.3.3.5 Receive Bit Field Extract Control Register (RBIFX)

The RBIFX register provides a set of four 6-bit offsets for locating up to four octets in a received frame and passing them to the receive queue filer as the user-defined ARB property. Through RBIFX a custom ARB filer property can be constructed from arbitrary bytes, which allows frame filing on the basis of bitfields not ordinarily provided to the filer, such as bits from the Ethernet preamble or TCP flags. The value of property ARB is the concatenation of {B0, B1, B2, B3} to 32-bits, where B0–B3 are the bytes as defined by RBIFX.

Figure 14-28 describes the definition for the RBIFX register.Note: when the eTSEC is configured to receive frame through the FIFO packet interface, a value of B*n*CTL = 01 is not supported unless RCTRL[PRSFM]=1 and RCTRL[PRSDEP] is configured to parse L2 packets over the FIFO interface. Below is a list of arbitrary extraction requirements:

- Byte extraction level cannot exceed the parser depth: a value of B*n*CTL=10 requires RCTRL[PRSDEP]=1x and a value of B*n*CTL=11 requires RCTRL[PRSDEP]=11.
- For B*n*CTL = 01, B*n*OFFSET = 7 is not supported.
- For values of B*n*CTL=10 or B*n*CTL=11, the controller extracts the defined bytes even if it does not recognize the L3 or L4 header, respectively.
- No L4 extraction is done if a packet is an IPV4 or IPV6 fragment frame.
- If no extraction occurs due to B*n*OFFSET longer than frame data or it is an unsupported B*n*OFFSET, the B*n* extraction values are filled with zeros.

Offset eTSEC1:0x2_4330;
eTSEC3:0x2_6330

Access: Read/Write

| | 0 1 | 2 7 | 8 9 | 10 15 | 16 17 | 18 23 | 24 25 | 26 31 |
|---|---|---|---|---|---|---|---|---|
| R<br>W | B0CTL | B0OFFSET | B1CTL | B1OFFSET | B2CTL | B2OFFSET | B3CTL | B3OFFSET |
| Reset | All zeros | | | | | | | |

**Figure 14-28. RBIFX Register Definition**

Table 14-33 describes the RBIFX register.

**Table 14-33. RBIFX Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–1 | B0CTL | Location of byte 0 of property ARB.<br>00 Byte 0 is not extracted, and appears as zero in property ARB.<br>01 Byte 0 is located in the received frame at offset (B0OFFSET – 8) bytes from the first byte of the Ethernet DA. In non-FIFO modes, a negative effective offset points to bytes of the standard Ethernet preamble. Values of B0OFFSET less than 8 are reserved in FIFO modes.<br>10 Byte 0 is located in the received frame at offset B0OFFSET bytes from the byte after the last byte of the layer 2 header.<br>11 Byte 0 is located in the received frame at offset B0OFFSET bytes from the byte after the last byte of the layer 3 header. |
| 2–7 | B0OFFSET | Offset relative to the header defined by B0CTL that locates byte 0 of property ARB. An effective offset of zero points to the first byte of the specified header. |
| 8–9 | B1CTL | Location of byte 1 of property ARB.<br>00 Byte 1 is not extracted, and appears as zero in property ARB.<br>01 Byte 1 is located in the received frame at offset (B1OFFSET – 8) bytes from the first byte of the Ethernet DA. In non-FIFO modes, a negative effective offset points to bytes of the standard Ethernet preamble. Values of B1OFFSET less than 8 are reserved in FIFO modes.<br>10 Byte 0 is located in the received frame at offset B1OFFSET bytes from the byte after the last byte of the layer 2 header.<br>11 Byte 0 is located in the received frame at offset B1OFFSET bytes from the byte after the last byte of the layer 3 header. |
| 10–15 | B1OFFSET | Offset relative to the header defined by B1CTL that locates byte 1 of property ARB. An effective offset of zero points to the first byte of the specified header. |
| 16–17 | B2CTL | Location of byte 2 of property ARB.<br>00 Byte 2 is not extracted, and appears as zero in property ARB.<br>01 Byte 2 is located in the received frame at offset (B2OFFSET – 8) bytes from the first byte of the Ethernet DA. In non-FIFO modes, a negative effective offset points to bytes of the standard Ethernet preamble. Values of B2OFFSET less than 8 are reserved in FIFO modes.<br>10 Byte 0 is located in the received frame at offset B2OFFSET bytes from the byte after the last byte of the layer 2 header.<br>11 Byte 0 is located in the received frame at offset B2OFFSET bytes from the byte after the last byte of the layer 3 header. |
| 18–23 | B2OFFSET | Offset relative to the header defined by B2CTL that locates byte 2 of property ARB. An effective offset of zero points to the first byte of the specified header. |

**Table 14-33. RBIFX Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 24–25 | B3CTL | Location of byte 3 of property ARB.<br>00 Byte 3 is not extracted, and appears as zero in property ARB.<br>01 Byte 3 is located in the received frame at offset (B3OFFSET – 8) bytes from the first byte of the Ethernet DA. In non-FIFO modes, a negative effective offset points to bytes of the standard Ethernet preamble. Values of B3OFFSET less than 8 are reserved in FIFO modes.<br>10 Byte 0 is located in the received frame at offset B3OFFSET bytes from the byte after the last byte of the layer 2 header.<br>11 Byte 0 is located in the received frame at offset B3OFFSET bytes from the byte after the last byte of the layer 3 header. |
| 26–31 | B3OFFSET | Offset relative to the header defined by B3CTL that locates byte 3 of property ARB. An effective offset of zero points to the first byte of the specified header. |

### 14.5.3.3.6 Receive Queue Filer Table Address Register (RQFAR)

RQFAR, shown in Figure 14-29, contains the index of the current, indirectly accessible entry of the received queue filer table. Each table entry occupies a pair of 32-bit words, denoted RQCTRL and RQPROP. To access the RQCTRL and RQPROP words of entry *n*, write *n* to RQFAR. Then read or write the indexed RQCTRL and RQPROP words by reading or writing the RQFCR and RQFPR registers, respectively.

Offset eTSEC1:0x2_4334;
eTSEC3:0x2_6334

Access: Read/Write

| | 0 … 23 | 24 … 31 |
|---|---|---|
| R<br>W | — | RQFAR |
| Reset | All zeros | |

**Figure 14-29. Receive Queue Filer Table Address Register Definition**

Table 14-34 describes the fields of the RQFAR register.

**Table 14-34. RQFAR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–23 | — | Reserved |
| 24–31 | RQFAR | Current index of receive queue filer table, which spans a total of 256 entries. |

### 14.5.3.3.7 Receive Queue Filer Table Control Register (RQFCR)

RQFCR is accessed to read or write the RQCTRL words in entries of the receive queue filer table. The table entries are described in greater detail in Section 14.6.5.2, “Receive Queue Filer.” The word accessed through RQFCR is defined by the current value of RQFAR.

Figure 14-30 describes the definition for the RQFCR register.

Offset eTSEC1:0x2_4338;
eTSEC3:0x2_6338

Access: Read/Write

| | 0 | 1 | 15 | 16 | 21 | 22 | 23 | 24 | 25 26 | 27 | 28 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | GPI | — | | Q | | CLE | REJ | AND | CMP | — | PID | |
| Reset | (undefined) | | | | | | | | | | | |

**Figure 14-30. Receive Queue Filer Table Control Register Definition**

Table 14-35 describes the fields of the RQFCR register.

**Table 14-35. RQFCR Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 0 | GPI | General purpose interrupt. When a property matches the value in the RQPROP entry at this index, and REJ = 0 and AND = 0, the filer will instruct the Rx descriptor controller to set IEVENT[FGPI] when the corresponding receive frame is written to memory.<br>If the timer is enabled (TMR_CTRL[TE] = 1), then TMR_PEVENT[RXP] will also be set. |
| 1–15 | — | Reserved, should be written with zero. |
| 16–21 | Q | Receive queue index, from 0 to 63, inclusive, written into the Rx frame control block associated with the received frame. When a property matches the value in the RQPROP entry at this index, and REJ = 0 and AND = 0, the frame is sent to either RxBD ring 0 (if RCTRL[FSQEN] = 1) or the RxBD ring with index (Q mod 8) and the filing table search is terminated. In the case where RCTRL[FSQEN] = 0, 8 virtual receive queues are overlaid on every RxBD ring, and software needs to consult the RQ field of the Rx frame control block to determine which virtual receive queue was chosen. |
| 22 | CLE | Cluster entry/exit (used in combination with AND bit). This bit brackets clusters, marking the start and end entries of a cluster. Clusters cannot be nested.<br>0 Regular RQCTRL entry.<br>1 If entry matches and AND = 1, treat subsequent entries as belonging to a nested cluster and enter the cluster; otherwise skip all entries up to and including the next cluster exit. If AND = 0, exit current cluster. |
| 23 | REJ | Reject frame. This bit and its specified action are ignored if AND = 1.<br>0 If entry matches, accept frame and file it to RxBD ring Q.<br>1 If entry matches, reject frame and discard it, ignoring Q. |
| 24 | AND | Match this entry and the next entry as a pair.<br>0 Match property[PID] against RQPROP, independent of the next entry.<br>1 Match property[PID] against RQPROP. If matched and CLE = 0, attempt to match next entry, otherwise, skip all entries up to and including the entry with AND = 0. If matched and CLE = 1, enter cluster of entries, otherwise, skip all entries up to and including the entry with CLE = 1 (cluster exit). |

**Table 14-35. RQFCR Field Descriptions (continued)**

| Bit | Name | Description |
|---|---|---|
| 25–26 | CMP | Comparison operation to perform on the RQPROP entry at this index when PID > 0. The property value extracted by the frame parser is masked by the 32-bit *mask_register* prior to comparison against RQPROP. However, the property value is not permanently altered by the value in *mask_register*. By default, *mask_register* is initialized to 0xFFFF_FFFF before each frame is processed.<br>In the case where PID = 0, CMP is interpreted as follows:<br>00/01 Filer *mask_register* is set to all 32 bits of RQPROP, and this entry always *matches*.<br>10/11 Filer *mask_register* is set to all 32 bits of RQPROP, and this entry always *fails to match*.<br>In the case where PID > 0, CMP is interpreted as follows (& is bit-wise AND operator):<br>00 *property*[PID] & *mask_register* = RQPROP<br>01 *property*[PID] & *mask_register* >= RQPROP<br>10 *property*[PID] & *mask_register* != RQPROP<br>11 *property*[PID] & *mask_register* < RQPROP |
| 27 | — | Reserved, should be written with zero. |
| 28–31 | PID | Property identifier. The value in the RQPROP entry at this index is interpreted according to PID (see Table 14-36). |

### 14.5.3.3.8 Receive Queue Filer Table Property Register (RQFPR)

RQFPR (see Figure 14-31) is accessed to read or write the RQPROP words in entries of the receive queue filer table. The table entries are described in greater detail in Section 14.6.5.2, “Receive Queue Filer.” The word accessed through RQFPR is defined by the current value of RQFAR. Figure 14-31 and Figure 14-32 describe the fields of the RQFPR register according to property ID.

Offset eTSEC1:0x2_433C; eTSEC3:0x2_633C

Access: Read/Write

| | 0–31 |
|---|---|
| R<br>W | RQPROP |
| Reset | *(undefined)* |

**Figure 14-31. Receive Queue Filer Table Property IDs 0, 2–15 Register Definition**

Offset eTSEC1:0x2_433C;<br>eTSEC3:0x2_633C

Access: Read/Write

| | 0–13 | 14 | 15 |
|---|---|---|---|
| R<br>W | — | AR | ARQ |
| Reset | All zeros | | |

| | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28–29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | EBC | VLN | CFI | JUM | IPF | FIF | IP4 | IP6 | ICC | ICV | TCP | UDP | — | PER | EER |
| Reset | All zeros | | | | | | | | | | | | | | |

**Figure 14-32. Receive Queue Filer Table Property ID1 Register Definition**

Table 14-36 describes the fields of the RQFPR register.

**Table 14-36. RQFPR Field Descriptions**

| PID[1] | Bit | Name | Description |
|---|---|---|---|
| 0000 | 0–31 | MASK | Mask bits to be written to Filer *mask_register* for masking of property values. The rule match/fail status for this PID is determined by RQCTRL[CMP]. Since *mask_register* is bit-wise ANDed with properties, every bit of MASK that is cleared also results in the corresponding property bit being cleared in comparisons. Therefore setting MASK to 0xFFFF_FFFF ensures that all property bits participate in rule matches. |
| 0001 | 0–13 | — | Reserved |
| | 14 | AR | Set if an ARP response packet is seen. |
| | 15 | ARQ | Set if an ARP request packet is seen. |
| | 16 | EBC | Set if the destination Ethernet address is to the broadcast address. |
| | 17 | VLN | Set if a VLAN tag (Ethertype DFVLAN[TAG] or 0x8100) was seen in the frame. |
| | 18 | CFI | Set to the value of the Canonical Format Indicator in the VLAN control tag if VLAN is set, zero otherwise. |
| | 19 | JUM | Set if a jumbo Ethernet frame was parsed. |
| | 20 | IPF | Set if a fragmented IPv4 or IPv6 header was encountered.<br>See the descriptions of receive FCB fields IP and PRO in Section 14.6.4.3, “Receive Path Off-Load,” for more information on determining the status of received packets for which IPF is set. |
| | 21 | FIF | Set if the packet entered on eTSEC’s FIFO interface. |
| | 22 | IP4 | Set if an IPv4 header was parsed. |
| | 23 | IP6 | Set if an IPv6 header was parsed. |
| | 24 | ICC | Set if the IPv4 header checksum was checked. |
| | 25 | ICV | Set if the IPv4 header checksum was verified correct. |
| | 26 | TCP | Set if a TCP header was parsed. |
| | 27 | UDP | Set if a UDP header was parsed. |
| | 28–29 | — | Reserved. |
| | 30 | PER | Set on a parse error, such as header inconsistency. |
| | 31 | EER | Set on an Ethernet framing error that prevents parsing. |
| 0010 | 0–7 | ARB | User-defined arbitrary bit field property: byte 0 extracted. Defaults to 0x00. |
| | 8–15 | | User-defined arbitrary bit field property: byte 1 extracted. Defaults to 0x00. |
| | 16–23 | | User-defined arbitrary bit field property: byte 2 extracted. Defaults to 0x00. |
| | 24–31 | | User-defined arbitrary bit field property: byte 3 extracted. Defaults to 0x00. |
| 0011 | 0–7 | — | Reserved, should be written with zero. |
| | 8–31 | DAH | Destination MAC address, most significant 24 bits. Defaults to 0x000000. |
| 0100 | 0–7 | — | Reserved, should be written with zero. |
| | 8–31 | DAL | Destination MAC address, least significant 24 bits. Defaults to 0x000000. |

Table 14-36. RQFPR Field Descriptions (continued)

| PID[1] | Bit | Name | Description |
|---|---|---|---|
| 0101 | 0–7 | — | Reserved, should be written with zero. |
| | 8–31 | SAH | Source MAC address, most significant 24 bits. Defaults to 0x000000. |
| 0110 | 0–7 | — | Reserved, should be written with zero. |
| | 8–31 | SAL | Source MAC address, least significant 24 bits. Defaults to 0x000000. |
| 0111 | 0–15 | — | Reserved, should be written with zero. |
| | 16–31 | ETY | Ethertype of next layer protocol, that is, last ethertype if layer 2 headers nest. Defaults to 0xFFFF.<br>Using the filer to match ETY does not work in the case of PPPoE packets, because the PPPoE ethertype in the original packet, 0x8864, is always overwritten with the PPP protocol field. Thus, matches on ETY == 0x8864 always fail.<br>Instead, software should use PID=1 fields IP4 (ETY = 0x0021) and IP6 (ETY = 0x0057) to distinguish PPPoE session packets carrying IPv4 and IPv6 datagrams. Other PPP protocols are encoded in the ETY field, but many of them overlap with real ethertype definitions. Consult IANA and IEEE for possible ambiguities.<br>A value in the length/type field greater than 1500 and less than 1536 is treated as a type encoding by the parser. Since no recognized types exist in this range, the controller will not parse beyond the length/type field of any such frame.<br>Note that the eTSEC filer gets multiple packet attributes as a result of parsing the packet. The behavior of the eTSEC is that it pulls the innermost ethertype found in the packet; this means that in many supported protocols that have inner ethertypes, in order to file based on the outer ethertype, arbitrary extraction should be used instead of the ETY PID. There are four cases that need to be highlighted.<br>1. The jumbo ethertype (0x8870)—In this case, the eTSEC assumes that the following header is LLC/SNAP. LLC/SNAP has an associated Ethertype, and the ETY field is populated with that ethertype. This makes it impossible to file on jumbo frames.<br>In this case, one can use arbitrary extracted bytes to pull the outermost Ethertype.<br>2. The PPPoE ethertype described above.<br>3. The VLAN tag ethertype (0x8100)—In this case, one can use the PID=1 VLN bit to indicate that the packet had a VLAN tag.<br>4. The MPLS tagged packets. In this case, one can use arbitrary extraction bytes to compare to the actual ethertype if a filer rule is intending to file based on an MPLS label existence. |
| 1000 | 0–19 | — | Reserved, should be written with zero. |
| | 20–31 | VID | VLAN network identifier (as per IEEE Std 802.1Q). This value defaults to 0x000 if no VLAN tag was found, or the VLAN tag contained only priority information. |
| 1001 | 0–28 | — | Reserved, should be written with zero. |
| | 29–31 | PRI | VLAN user priority (as per IEEE Std 802.1p). This value defaults to 000 (best effort priority) if no VLAN tag was found. |
| 1010 | 0–23 | — | Reserved, should be written with zero. |
| | 24–31 | TOS | IPv4 header Type Of Service field or IPv6 Traffic Class field. This value defaults to 0x00 (default RFC 2474 best-effort behavior) if no IP header appeared.<br>Note that for IPv6 the Traffic Class field is extracted using the IP header definition in RFC 2460. IPv6 headers formed using the earlier RFC 1883 have a different format and must be handled with software. |
| 1011 | 0–23 | — | Reserved, should be written with zero. |
| | 24–31 | L4P | Layer 4 protocol identifier as per published IANA specification. This is the last recognized protocol type recognized in the case of IPv6 extension headers. This value defaults to 0xFF to indicate that no layer 4 header was recognized (possibly due to absence of an IP header). |

**Table 14-36. RQFPR Field Descriptions (continued)**

| PID[1] | Bit | Name | Description |
|---|---|---|---|
| 1100 | 0–31 | DIA | Destination IP address. If an IPv4 header was found, this is the entire destination address. If an IPv6 header was found, this is the 32 most significant bits of the 128-bit destination address. This value defaults to 0x0000_0000 if no IP header appeared. |
| 1101 | 0–31 | SIA | Source IP address. If an IPv4 header was found, this is the entire source address. If an IPv6 header was found, this is the 32 most significant bits of the 128-bit source address. This value defaults to 0x0000_0000 if no IP header appeared. |
| 1110 | 0–15 | — | Reserved, should be written with zero. |
|  | 16–31 | DPT | Destination port number for TCP or UDP headers. This value defaults to 0x0000 if no TCP or UDP headers were recognized. |
| 1111 | 0–15 | — | Reserved, should be written with zero. |
|  | 16–31 | SPT | Source port number for TCP or UDP headers. This value defaults to 0x0000 if no TCP or UDP headers were recognized. |

[1] PID is the property identifier field of the filer table control entry (see RQFCR[PID]) at the same index.

### 14.5.3.3.9 Maximum Receive Buffer Length Register (MRBLR)

The MRBLR register is written by the user. It informs the eTSEC how much space is in the receive buffer pointed to by the RxBD. Figure 14-33 describes the definition for the MRBLR.

Offset eTSEC1:0x2_4340;
eTSEC3:0x2_6340

Access: Read/Write

|  | 0–15 | 16–25 | 26–31 |
|---|---|---|---|
| R<br>W | — | MRBL | — |
| Reset | All zeros |  |  |

**Figure 14-33. MRBLR Register Definition**

**Table 14-37. MRBLR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–15 | — | Reserved |
| 16–25 | MRBL | Maximum receive buffer length. MRBL is the number of bytes that the eTSEC receiver writes to the receive buffer. The MRBL register is written by the user with a multiple of 64 for all modes. The eTSEC can write fewer bytes to the buffer than the value set in MRBL if a condition such as an error or end-of-frame occurs, but it never exceeds the MRBL value; therefore, user-supplied buffers must be at least as large as the MRBL. MRBL must be set, together with the number of buffer descriptors, to ensure adequate space for received frames. See Section 14.5.3.5.5, "Maximum Frame Length Register (MAXFRM)," for further discussion. |
| 26–31 | — | To ensure that MRBL is a multiple of 64, these bits are reserved and should be cleared. |

### 14.5.3.3.10 Receive Data Buffer Pointer High Register (RBDBPH)

The RBDBPH register is written by the user with the most significant address bits common to all RxBD buffer addresses, RxBD[Data Buffer Pointer]. As a consequence, Rx buffers must be placed in a 4 Gbyte segment of memory whose base address is prefixed by the bits in RBDBPH. The RxBD ring itself can reside in a different memory region (based at RBASEH). Figure 14-34 describes the definition for the RBDBPH register.

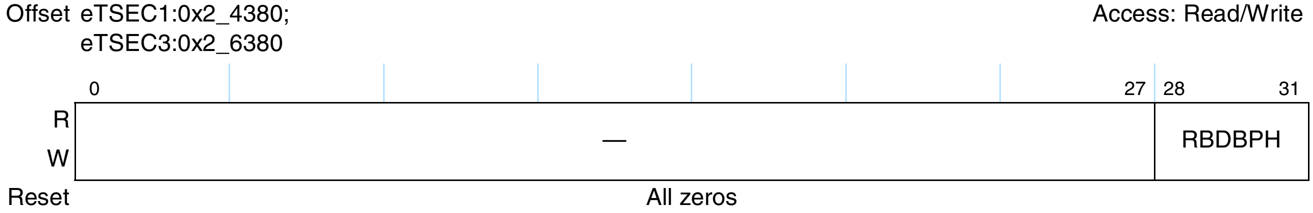


**Figure 14-34. RBDBPH Register Definition**

Table 14-38 describes the fields of the RBDBPH register.

**Table 14-38. RBDBPH Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–27 | — | Reserved |
| 28–31 | RBDBPH | Most significant bits common to all data buffer addresses contained in RxBDs. The user must initialize RBDBPH before enabling the eTSEC receive function. |

### 14.5.3.3.11 Receive Buffer Descriptor Pointers 0–7 (RBPTR0–RBPTR7)

RBPTR0–RBPTR7 each contains the low-order 32 bits of the next receive buffer descriptor address for their respective RxBD ring. Figure 14-35 describes the RBPTR registers. These registers takes on the value of their ring’s associated RBASE when the RBASE register is written by software. Software must not write RBPTR*n* while eTSEC is actively receiving frames. However, RBPTR*n* can be modified when the receiver is disabled or when no Rx buffer is in use (after a GRACEFUL STOP RECEIVE command is issued and the frame completes its reception) in order to change the next RxBD eTSEC receives.

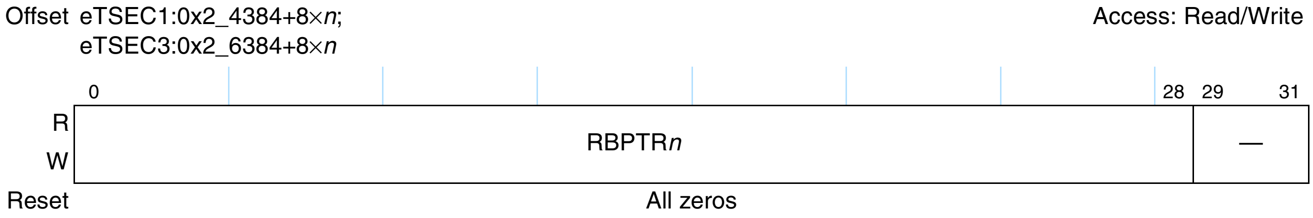


**Figure 14-35. RBPTR0–RBPTR7 Register Definition**

Table 14-24 describes the fields of the RBPTR*n* register.

**Table 14-39. RBPTR*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–28 | RBPTR*n* | Current RxBD pointer for RxBD ring *n*. Points to the current BD being processed or to the next BD the receiver uses when it is idling. After reset or when the end of the RxBD ring is reached, eTSEC initializes RBPTR*n* to the value in the corresponding RBASE*n*. The RBPTR register is internally written by the eTSEC’s DMA controller during reception. The pointer increments by 8 (bytes) each time a descriptor is closed successfully by the eTSEC. Note that the 3 least-significant bits of this register are read only and zero. |
| 29–31 | — | Reserved |

### 14.5.3.3.12 Receive Descriptor Base Address High Register (RBASEH)

The RBASEH register is written by the user with the most significant address bits common to all RxBD addresses, including RBASE0–RBASE7 and RBPTR0–RBPTR7. As a consequence, RxBD rings must be placed in a 4 Gbyte segment of memory whose base address is prefixed by the bits in RBASEH. However, Rx data buffers may potentially reside in a different memory region based at RBDBPH. Figure 14-36 describes the definition for the RBASEH register.

Offset eTSEC1:0x2_4400;
eTSEC3:0x2_6400

Access: Read/Write

| | 0–27 | 28–31 |
|---|---|---|
| R<br>W | — | RBASEH |
| Reset | All zeros | |

**Figure 14-36. RBASEH Register Definition**

Table 14-25 describes the fields of the RBASEH register.

**Table 14-40. RBASEH Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–27 | — | Reserved |
| 28–31 | RBASEH | Most significant bits common to all RxBD addresses—except data buffer pointers. The user must initialize RBASEH before enabling the eTSEC receive function. |

### 14.5.3.3.13 Receive Descriptor Base Address Registers (RBASE0–RBASE7)

The RBASE*n* registers are written by the user with the base address of each RxBD ring *n*. Each such value must be divisible by eight, since the 3 least-significant bits always write as 000. Figure 14-37 describes the RBASE*n* registers.

Offset eTSEC1:0x2_4404+8×*n*;
eTSEC3:0x2_6404+8×*n*

Access: Read/Write

| | 0–28 | 29–31 |
|---|---|---|
| R<br>W | RBASE*n* | — |
| Reset | All zeros | |

**Figure 14-37. RBASE Register Definition**

Table 14-26 describes the fields of the RBASE*n* registers.

**Table 14-41. RBASE0–RBASE7 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–28 | RBASE*n* | Receive base for ring *n*. RBASE defines the starting location in the memory map for the eTSEC RxBDs. This field must be 8-byte aligned. Together with setting the W (wrap) bit in the last BD, the user can select how many BDs to allocate for the receive packets. The user must initialize RBASE before enabling the eTSEC receive function on the associated ring. |
| 29–31 | — | Reserved |

#### 14.5.3.3.14 Receive Stamp Register (TMR_RXTS_H/L)

Receive time stamp register (RXTS_H/L). This register holds the value present in TMR_CNT_H/L when the eTSEC detects a new incoming Ethernet frame. This register is only updated when the precision time stamp logic is enable via TMR_CTRL[TE]. This register is read only in normal operation. Figure 14-38 describes the definition for the RXTS_H/L register.

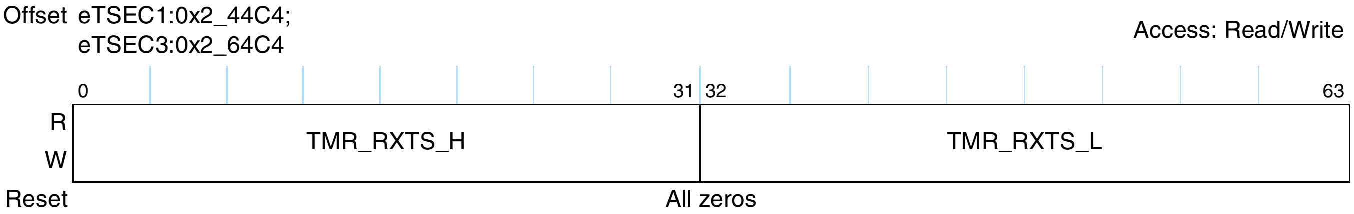


**Figure 14-38. TMR_RXTS_H/L Register Definition**

Table 14-42 describes the fields of the TMR_RXTS_H/L register.

**Table 14-42. TMR_RXTS_H/L Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–63 | TMR_RXTS_H/L | Value of the eTSEC precision timer upon detection of a start of frame symbol for the received frame. |

### 14.5.3.4 MAC Functionality

This section describes the MAC registers and provides a brief overview of the functionality that can be exercised through the use of these registers, particularly those that provide functionality not explicitly required by the IEEE 802.3 standard. All of the MAC registers are 32 bits wide.

#### 14.5.3.4.1 Configuring the MAC

The MAC configuration registers 1 and 2 provide for configuring the MAC in multiple ways:

- Adjusting the preamble length—The length of the preamble can be adjusted from the nominal seven bytes to some other (non-zero) value. Should custom preamble insertion/extraction be configured, then this register must by left at its default value.
- Varying pad/CRC combinations—Three different pad/CRC combinations are provided to handle a variety of system requirements. Simplest are frames that already have a valid frame check sequence (FCS) field. The other two options include appending a valid CRC or padding and then appending a valid CRC, resulting in a minimum frame of 64 octets. In addition to the programmable register set, the pad/CRC behavior can be dynamically adjusted on a per-packet basis.

#### 14.5.3.4.2 Controlling CSMA/CD

The half-duplex register (HAFDUP) allows control over the carrier-sense multiple access/collision detection (CSMA/CD) logic of the eTSEC. Half-duplex mode is only supported for 10- and 100-Mbps operation. Following the completion of the packet transmission the part begins timing the inter packet gap (IPG) as programmed in the back-to-back IPG configuration register. The system is now free to begin another frame transfer.

In full-duplex mode both the carrier sense (CRS) and collision (COL) indications from the PHY are ignored, but in half-duplex mode the eTSEC defers to CRS, and following a carrier event, times the IPG using the non-back-to-back IPG configuration values that include support for the optional two-thirds/one-third CRS deferral process. This optional IPG mechanism enhances system robustness and ensures fair access to the medium. During the first two-thirds of the IPG, the IPG timer is cleared if CRS is sensed. During the final one-third of the IPG, CRS is ignored and the transmission begins once IPG is timed. The two-thirds/one-third ratio is the recommended value.

### 14.5.3.4.3 Handling Packet Collisions

While transmitting a packet in half-duplex mode, the eTSEC is sensitive to COL. (Note that in RGMII and SGMII, there is no COL/CRS. Instead, COL and CRS are derived from the equivalents of RX_DV and TX_EN.) If a collision occurs, it aborts the packet and outputs the 32-bit jam sequence. The jam sequence is comprised of several bits of the CRC, inverted to guarantee an invalid CRC upon reception. A signal is sent to the system indicating that a collision occurred and that the start of the frame is needed for retransmission. The eTSEC then backs off of the medium for a time determined by the truncated binary exponential back off (BEB) algorithm. Following this back-off time, the packet is retried. The back-off time can be skipped if configured through the half-duplex register. However, this is non-standard behavior and its use must be carefully applied. Should any one packet experience excessive collisions, the packet is aborted. The system should flush the frame and move to the next one in line. If the system requests to send a packet while the eTSEC is deferring to a carrier, the eTSEC simply waits until the end of the carrier event and the timing of IPG before it honors the request.

If packet transmission attempts experience collisions, the eTSEC outputs the jam sequence and waits some amount of time before retrying the packet. This amount of time is determined by a controlled randomization process called truncated binary exponential back-off. The amount of time is an integer number of slot times. The number of slot times to delay before the *n*th retransmission attempt is chosen as a uniformly-distributed random integer *r* in the range:

$$0 \le r \le 2^{\mathrm{k}}, \text{ where } \mathrm{k} = \min(n,10).$$

So after the first collision, the eTSEC backs off either 0 or 1 slot times. After the fifth collision, the eTSEC backs off between 0 and 32 slot times. After the tenth collision, the maximum number of slot times to back off is 1024. This can be adjusted through the half-duplex register. An alternate truncation point, such as 7 for instance, can be programmed. On average, the MAC is more aggressive after seven collisions than other stations on the network.

### 14.5.3.4.4 Controlling Packet Flow

Packet flow can be dealt with in a number of ways within eTSEC. A default retransmit attempt limit of 15 can be reduced using the half-duplex register. The slot time or collision window can be used to gate the retry window and possibly reduce the amount of transmit buffering within the system. The slot time for 10/100 Mbps is 512 bit times. Because the slot time begins at the beginning of the packet (including preamble), the end occurs around the 56th byte of the frame data. Slot time in 1000-Mbps mode is not supported.

Full-duplex flow control is provided for in IEEE 802.3x. Currently the standard does not address flow control in half-duplex environments. Common in the industry, however, is the concept of back pressure.

The eTSEC implements the optional back pressure mechanism using the raise carrier method. If the system receive logic wishes to stop the reception of packets in a network-friendly way, transmit half-duplex flow control (THDF) is set (TCTRL[THDF]). If the medium is idle, the eTSEC raises carrier by transmitting preamble. Other stations on the half-duplex network then defer to the carrier.

In the event the preamble transmission happens to cause a collision, the eTSEC ensures the minimum 96-bit presence on the wire, then drops preamble and waits a back-off time depending on the value of the back-pressure-no-back-off configuration bit HAFDUP[BP No BackOff]. These transmitting-preamble-for-back pressure collisions are not counted. If HAFDUP[BP No BackOff] is set, the eTSEC waits an inter-packet gap before resuming the transmission of preamble following the collision and does not defer. If HAFDUP[BP No BackOff] is cleared, the eTSEC adheres to the truncated BEB algorithm that allows the possibility of packets being received. This also can be detrimental in that packets can now experience excessive collisions, causing them to be dropped in the stations from which they originate. To reduce the likelihood of lost packets and packets leaking through the back pressure mechanism, HAFDUP[BP No BackOff] must be set.

The eTSEC drops carrier (cease transmitting preamble) periodically to avoid excessive defer conditions in other stations on the shared network. If, while applying back pressure, the eTSEC is requested to send a packet, it stops sending preamble, and waits one IPG before sending the packet. HAFDUP[BP No BackOff] applies for any collision that occurs during the sending of this packet. Collisions for packets while half duplex back pressure is asserted are counted. The eTSEC does not defer while attempting to send packets while in back pressure. Again, back pressure is non-standard, yet it can be effective in reducing the flow of receive packets.

#### 14.5.3.4.5 Controlling PHY Links

Control and status to and from the PHY is provided through the two-wire MII management interface described in IEEE 802.3u. The MII management registers (MII management configuration, command, address, control, status, and indicator registers) are used to exercise this interface between a host processor and one or more PHY devices (including the TBI).

The eTSEC MII’s registers provide the ability to perform continuous read cycles (called a scan cycle); although, scan cycles are not explicitly defined in the standard. If requested (by setting MIIMCOM[Scan Cycle]), the part performs repetitive read cycles of the PHY status register, for example. In this way, link characteristics may be monitored more efficiently. The different fields in the MII management indicator register (scan, not valid and busy) are used to indicate availability of each read of the scan cycle to the host from MIIMSTAT[PHY scan].

Yet another parameter that can be modified through the MII registers is the length of the MII management interface preamble. After establishing that a PHY supports preamble suppression, the host may so configure the eTSEC. While enabled, the length of MII management frames are reduced from 64 clocks to 32 clocks. This effectively doubles the efficiency of the interface.

### 14.5.3.5 MAC Registers

This section describes the MAC registers.

#### 14.5.3.5.1 MAC Configuration 1 Register (MACCFG1)

MACCFG1 is written by the user. Figure 14-39 describes the definition for the MACCFG1 register.

Offset eTSEC1:0x2_4500;
eTSEC3:0x2_6500

Access: Mixed

| | 0 | 1–11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|
| R | Soft_Reset | — | Reset Rx MC | Reset Tx MC | Reset Rx Fun | Reset Tx Fun |
| W | | | | | | |
| Reset | All zeros | | | | | |

| | 16–22 | 23 | 24–25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|
| R | — | Loop Back | — | Rx_Flow | Tx_Flow | Sync'd Rx EN | Rx_EN | Sync'd Tx EN | Tx_EN |
| W | | | | | | | | | |
| Reset | All zeros | | | | | | | | |

**Figure 14-39. MACCFG1 Register Definition**

Table 14-43 describes the fields of the MACCFG1 register.

**Table 14-43. MACCFG1 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | Soft_Reset | Soft reset. This bit is cleared by default. See Section 14.6.3.2, "Soft Reset and Reconfiguring Procedure," for more information on setting this bit.<br>0 Normal operation.<br>1 Place the entire MAC in reset except for the host interface. |
| 1–11 | — | Reserved |
| 12 | Reset Rx MC | Reset receive MAC control block. This bit is cleared by default.<br>0 Normal operation.<br>1 Place the receive part of the MAC in reset. This block detects control frames and contains the pause timers. |
| 13 | Reset Tx MC | Reset transmit MAC control block. This bit is cleared by default.<br>0 Normal operation.<br>1 Place the transmit part of the MAC in reset. This block multiplexes data and control frame transfers. It also responds to XOFF PAUSE control frames. |
| 14 | Reset Rx Fun | Reset receive function block. This bit is cleared by default.<br>0 Normal operation.<br>1 Place the receive function in reset. This block performs the receive frame protocol. |
| 15 | Reset Tx Fun | Reset transmit function block. This bit is cleared by default.<br>0 Normal operation.<br>1 Place the transmit function in reset. This block performs the frame transmission protocol. |
| 16–22 | — | Reserved |

**Table 14-43. MACCFG1 Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 23 | Loop Back | Loop back. This bit is cleared by default.<br>0 Normal operation.<br>1 Loop back the MAC transmit outputs to the MAC receive inputs. |
| 24–25 | — | Reserved |
| 26 | Rx_Flow | Receive flow. This bit is cleared by default.<br>Must be 0 if MACCFG2[Full Duplex] = 0.<br>0 The receive MAC control ignores PAUSE flow control frames.<br>1 The receive MAC control detects and acts on PAUSE flow control frames.<br>**Note:** Should not be set when operating in Half-Duplex mode |
| 27 | Tx_Flow | Transmit flow. This bit is cleared by default.<br>Must be 0 if MACCFG2[Full Duplex] = 0.<br>0 The transmit MAC control may not send PAUSE flow control frames if requested by the system.<br>**Note:** 1The transmit MAC control may send PAUSE flow control frames if requested by the system.Should not be set when operating in Half-Duplex mode |
| 28 | Sync'd Rx EN | Receive enable synchronized to the receive stream. (Read-only)<br>0 Frame reception is not enabled.<br>1 Frame reception is enabled. |
| 29 | Rx_EN | Receive enable. This bit is cleared by default. If set, prior to clearing this bit, set DMACTRL[GRS] then confirm subsequent occurrence of the graceful receive stop interrupt (IEVENT[GRSC] is set).<br>0 The MAC may not receive frames from the PHY.<br>1 The MAC may receive frames from the PHY. |
| 30 | Sync'd Tx EN | Transmit enable synchronized to the transmit stream. (Read-only)<br>0 Frame transmission is not enabled.<br>1 Frame transmission is enabled. |
| 31 | Tx_EN | Transmit enable. This bit is cleared by default. If set, prior to clearing this bit, set DMACTRL[GTS] then confirm subsequent occurrence of the graceful receive stop interrupt (IEVENT[GTSC] is set).<br>0 The MAC may not transmit frames from the system.<br>1 The MAC may transmit frames from the system. |

#### 14.5.3.5.2 MAC Configuration 2 Register (MACCFG2)

The MACCFG2 register is written by the user. Figure 14-40 describes the definition for the MACCFG2 register.

Offset eTSEC1:0x2_4504;
eTSEC3:0x2_6504

Access: Read/Write

| | 0–15 |
|---|---|
| R<br>W | — |
| Reset | All zeros |

| | 16–19 | 20–21 | 22–23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Preamble Length | — | I/F Mode | PreAmRxEN | PreAmTxEN | Huge Frame | Length check | MPEN | PAD/CRC | CRC EN | Full Duplex |
| Reset | 0 1 1 1 | 0 0 | 0 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Figure 14-40. MACCFG2 Register Definition**

Table 14-44 describes the fields of the MACCFG2 register.

**Table 14-44. MACCFG2 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–15 | — | Reserved |
| 16–19 | Preamble Length | This field determines the length in bytes of the preamble field preceding each Ethernet start-of-frame delimiter byte. Values from 0x3 to 0xF are supported by the controller. The default value of 0x7 should not be altered in order to guarantee reliable operation with IEEE 802.3 compliant hardware. |
| 20–21 | — | Reserved |
| 22–23 | I/F Mode | This field determines the type of interface to which the MAC is connected. Its default is 00.<br>00 Reserved bit mode (not supported) (10 Mbps GENDEC/GPSI)<br>01 Nibble mode (MII) (10/100 Mbps MII/RMII)<br>10 Byte mode (GMII/TBI) (1000 MbpsGMII/TBI). Reserved if neither GMII or TBI are supported.<br>11 Reserved |
| 24 | PreAM RxEN | User defined preamble enable for received frames. This bit is cleared by default.<br>0 The MAC skips the Ethernet preamble without returning it.<br>1 The MAC recovers the received Ethernet preamble and passes it to the driver at the start of each received frame. If the preamble is less than 7 bytes, 0's are prepended to pad it to 7 bytes. Not applicable to FIFO or RMII 10/100 modes. |
| 25 | PreAM TxEN | User defined preamble enable for transmitted frames. This bit is cleared by default.<br>0 The MAC generates a standard Ethernet preamble.<br>1 If a user-defined preamble has been passed to the MAC it is transmitted instead of the standard preamble. Otherwise the standard Ethernet preamble is generated. The Preamble Length field should be left at its default setting if a user-defined preamble is transmitted. Not applicable to FIFO or RMII 10/100 modes. |

**Table 14-44. MACCFG2 Field Descriptions (continued)**

<table>
<tr><th>Bits</th><th>Name</th><th>Description</th></tr>
<tr><td>26</td><td>Huge Frame</td><td>Huge frame enable. This bit is cleared by default.<br>0 Limit the length of frames received to less than or equal to the maximum frame length value (MAXFRM[Maximum Frame]) and limit the length of frames transmitted to less than the maximum frame length.<br>See Section 14.6.8, “Buffer Descriptors,” for further details of buffer descriptor bit updating.
<table>
<tr><th>Frame type</th><th>Frame length</th><th>Packet truncation</th><th>Buffer descriptor updated</th></tr>
<tr><td>Receive or transmit</td><td>> maximum frame length</td><td>yes</td><td>yes</td></tr>
<tr><td>Receive</td><td>= maximum frame length</td><td>no</td><td>no</td></tr>
<tr><td>Transmit</td><td>= maximum frame length</td><td>no</td><td>yes</td></tr>
<tr><td>Receive or transmit</td><td>< maximum frame length</td><td>no</td><td>no</td></tr>
</table>
1 Frames are transmitted and received regardless of their relationship to the maximum frame length.<br>Note that if Huge Frame is cleared, the user must ensure that adequate buffer space is allocated for received frames. See Section 14.5.3.5.5, “Maximum Frame Length Register (MAXFRM),” for further information.</td></tr>
<tr><td>27</td><td>Length check</td><td>Length check. This bit is cleared by default.<br>0 No length field checking is performed.<br>1 The MAC checks the frame’s length field on receive to ensure it matches the actual data field length. Transmitted frames are not checked.</td></tr>
<tr><td>28</td><td>MPEN</td><td>Magic packet enable for Ethernet modes. This bit is cleared by default. MPEN should be enabled only after GRACEFUL RECEIVE STOP and GRACEFUL TRANSMIT STOP are completed successfully (in other words, transmission and reception have stopped).<br>0 Normal receive behavior on receive, or Magic Packet mode has exited with reception of a valid Magic Packet.<br>1 Commence Magic Packet detection by the MAC provided that frame reception is enabled in MACCFG1. In this mode the MAC ignores all received frames until the specific Magic Packet frame is received, at which point this bit is cleared by the eTSEC, and a maskable interrupt through IEVENT[MAG] occurs.</td></tr>
<tr><td>29</td><td>PAD/CRC</td><td>Pad and append CRC. This bit is cleared by default.This bit must be set when in half-duplex mode (MACCFG2[Full Duplex] is cleared).<br>0 Frames presented to the MAC have a valid length and contain a CRC.<br>1 The MAC pads all transmitted short frames and appends a CRC to every frame regardless of padding requirement.</td></tr>
<tr><td>30</td><td>CRC EN</td><td>CRC enable. If the configuration bit PAD/CRC ENABLE or the per-packet PAD/CRC ENABLE is set, CRC ENABLE is ignored. This bit is cleared by default.<br>0 Frames presented to the MAC have a valid length and contain a valid CRC.<br>1 The MAC appends a CRC on all frames. Clear this bit if frames presented to the MAC have a valid length and contain a valid CRC.</td></tr>
<tr><td>31</td><td>Full Duplex</td><td>Full duplex configure. This bit is cleared by default.<br>0 The MAC operates in half-duplex mode only.<br>1 The MAC operates in full-duplex mode.</td></tr>
</table>

### 14.5.3.5.3 Inter-Packet Gap/Inter-Frame Gap Register (IPGIFG)

The IPGIFG register is written by the user. Figure 14-41 describes the definition for IPGIFG.

Offset eTSEC1:0x2_4508;
eTSEC3:0x2_6508

Access: Read/Write

| | 0 | 1–7 | 8 | 9–15 | 16–23 | 24 | 25–31 |
|---|---|---|---|---|---|---|---|
| R<br>W | — | Non-Back-to-Back Inter-Packet-Gap, Part 1 | — | Non-Back-to-Back Inter-Packet-Gap, Part 2 | Minimum IFG Enforcement | — | Back-to-Back Inter-Packet-Gap |
| Reset | 0 | 1 0 0 0 0 0 0 | 0 | 1 1 0 0 0 0 0 | 0 1 0 1 0 0 0 0 | 0 | 1 1 0 0 0 0 0 |

**Figure 14-41. IPGIFG Register Definition**

Table 14-45 describes the fields of the IPGIFG register.

**Table 14-45. IPGIFG Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | — | Reserved |
| 1–7 | Non-Back-to-Back Inter-Packet-Gap, Part 1 | This is a programmable field representing the optional carrier sense window referenced in IEEE 802.3/4.2.3.2.1 ‘carrier deference’. If carrier is detected during the timing of IPGR1, the MAC defers to carrier. If, however, carrier becomes active after IPGR1, the MAC continues timing IPGR2 and transmits, knowingly causing a collision, thus ensuring fair access to medium. Its range of values is 0x00 to IPGR2. Its default is 0x40 (64d) which follows the two-thirds/one-third guideline. |
| 8 | — | Reserved |
| 9–15 | Non-Back-to-Back Inter-Packet-Gap, Part 2 | This is a programmable field representing the non-back-to-back inter-packet-gap in bits. Its default is 0x60 (96d), which represents the minimum IPG of 96 bits. |
| 16–23 | Minimum IFG Enforcement | This is a programmable field representing the minimum number of bits of IFG to enforce between frames. A frame is dropped whose IFG is less than that programmed. The default setting of 0x50 (80d) represents half of the nominal minimum IFG which is 160 bits. |
| 24 | — | Reserved |
| 25–31 | Back-to-Back Inter-Packet-Gap | This is a programmable field representing the IPG between back-to-back packets. This is the IPG parameter used exclusively in full-duplex mode and in half-duplex mode if two transmit packets are sent back-to-back. Set this field to the number of bits of IPG desired. The default setting of 0x60 (96d) represents the minimum IPG of 96 bits. |

### 14.5.3.5.4 Half-Duplex Register (HAFDUP)

The HAFDUP register is written by the user. Figure 14-42 describes the HAFDUP register.

Offset eTSEC1:0x2_450C;
eTSEC3:0x2_650C

Access: Read/Write

| | 0–7 | 8–11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|
| R<br>W | — | Alternate BEB Truncation | Alt BEB | BP No BackOff | No BackOff | Excess Defer |
| Reset | 0 0 0 0 0 0 0 0 | 1 0 1 0 | 0 | 0 | 0 | 1 |

| | 16–19 | 20–25 | 26–31 |
|---|---|---|---|
| R<br>W | Retransmission Maximum | — | Collision Window |
| Reset | 1 1 1 1 | 0 0 0 0 0 0 | 1 1 0 1 1 1 |

**Figure 14-42. Half-Duplex Register Definition**

Table 14-46 describes the fields of the HAFDUP register.

**Table 14-46. HAFDUP Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–7 | — | Reserved |
| 8–11 | Alternate BEB Truncation | This field is used while ALTERNATE BINARY EXPONENTIAL BACKOFF ENABLE is set. The value programmed is substituted for the Ethernet standard value of ten. Its default is 0xA. |
| 12 | Alt BEB | Alternate binary exponential backoff. This bit is cleared by default.<br>0 The Tx MAC follows the standard binary exponential back off rule.<br>1 The Tx MAC uses the ALTERNATE BINARY EXPONENTIAL BACKOFF TRUNCATION setting instead of the 802.3 standard tenth collision. The standard specifies that any collision after the tenth uses one less than 210 as the maximum backoff time. |
| 13 | BP No BackOff | Back pressure no backoff. This bit is cleared by default.<br>0 The Tx MAC follows the binary exponential back off rule.<br>1 The Tx MAC immediately re-transmits, following a collision, during back pressure operation. |
| 14 | No BackOff | No backoff. This bit is cleared by default.<br>0 The Tx MAC follows the binary exponential back off rule.<br>1 The Tx MAC immediately re-transmits following a collision. |
| 15 | Excess Defer | Excessively deferred. This bit is set by default.<br>0 The Tx MAC aborts the transmission of a packet that is excessively deferred.<br>1 The Tx MAC allows the transmission of a packet that is excessively deferred. |
| 16–19 | Retransmission Maximum | This is a programmable field specifying the number of retransmission attempts following a collision before aborting the packet due to excessive collisions. The standard specifies the attempt limit to be 0xF (15d). Its default value is 0xF. |
| 20–25 | — | Reserved |
| 26–31 | Collision Window | This is a programmable field representing the slot time or collision window during which collisions occur in properly configured networks. Because the collision window starts at the beginning of transmission, the preamble and SFD are included. Its default of 0x37 (55d) corresponds to the count of frame bytes at the end of the window. |

### 14.5.3.5.5 Maximum Frame Length Register (MAXFRM)

The MAXFRM register is written by the user. Figure 14-43 shows the MAXFRM register.

Offset eTSEC1:0x2_4510;
eTSEC3:0x2_6510

Access: Read/Write

| | 0 | 15 | 16 | 31 |
|---|---|---|---|---|
| R<br>W | — | | Maximum Frame | |
| Reset | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | | 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 | |

**Figure 14-43. Maximum Frame Length Register Definition**

Table 14-47 describes the fields of the MAXFRM register.

**Table 14-47. MAXFRM Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–15 | — | Reserved |
| 16–31 | Maximum Frame | This field is set to 0x0600 (1536 bytes) by default and always must be set to a value greater than or equal to 0x0040 (64 bytes), but not greater than 0x2580 (9600 bytes). It sets the maximum Ethernet frame size in both the transmit and receive directions. (Refer to MACCFG2[Huge Frame].) It does not affect the size of packets sent or received via the FIFO packet interface.<br>Note that if MACCFG2[Huge Frame] = 0, the value of this field must be less than or equal to MRBLR[MRBL] × (minimum number of RxBDs per ring). See Section 14.5.3.5.2, “MAC Configuration 2 Register (MACCFG2),” Section 14.5.3.3.9, “Maximum Receive Buffer Length Register (MRBLR),” and Section 14.6.8.3, “Receive Buffer Descriptors (RxBD).” |

### 14.5.3.5.6 MII Management Configuration Register (MIIMCFG)

The MIIMCFG register is written by the user to configure all MII management operations. Note that MII management hardware is shared by all eTSECs. Thus, only through the MIIM registers of eTSEC1 can external PHYs be accessed and configured. Note: when an eTSEC is configured to use TBI/RTBI, configuration of the TBI/RTBI (described in Section 14.5.4, “Ten-Bit Interface (TBI)”) is done through the MIIM registers for that eTSEC. For example, if a TBI/RTBI interface is required on eTSEC2, then the MIIM registers starting at offset 0x2_5520 are used to configure it.

Figure 14-44 describes the definition for the MIIMCFG register.

Offset eTSEC1:0x2_4520

Access: Read/Write

| | 0 | 1 | 26 | 27 | 28 | 29 | 31 |
|---|---|---|---|---|---|---|---|
| R<br>W | Reset Mgmt | — | | No Pre | — | MgmtClk | |
| Reset | 0 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | | 0 | 0 | 1 1 1 | |

**Figure 14-44. MII Management Configuration Register Definition**

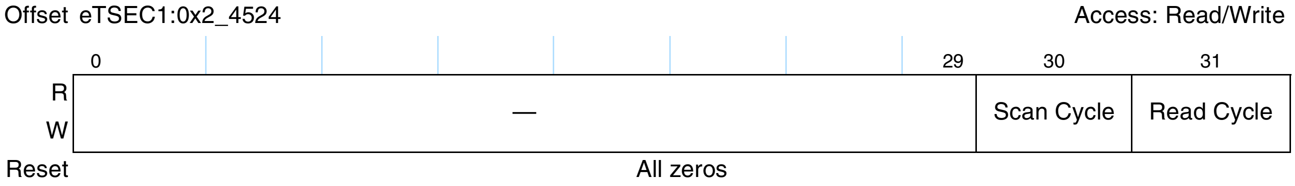
Table 14-48 describes the fields of the MIIMCFG register.

**Table 14-48. MIIMCFG Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | Reset Mgmt | Reset management. This bit is cleared by default.<br>0 Allow the MII MGMT to perform mgmt read/write cycles if requested through the host interface.<br>1 Reset the MII MGMT. |
| 1–26 | — | Reserved |
| 27 | No Pre | Preamble suppress. This bit is cleared by default.<br>0 The MII MGMT performs Mgmt read/write cycles with 32 clocks of preamble.<br>1 The MII MGMT suppresses preamble generation and reduces the Mgmt cycle from 64 clocks to 32 clocks. This is in accordance with IEEE 802.3/22.2.4.4.2. |
| 28 | — | Reserved |
| 29–31 | MgmtClk | This field determines the clock frequency of the MII management clock (EC_MDC). Its default value is 111.<br>**Note:** The eTSEC system clock is derived from (CCB Clock)/2.<br>000 1/4 of the eTSEC system clock divided by 8<br>001 1/4 of the eTSEC system clock divided by 8<br>010 1/6 of the eTSEC system clock divided by 8<br>011 1/8 of the eTSEC system clock divided by 8<br>100 1/10 of the eTSEC system clock divided by 8<br>101 1/14 of the eTSEC system clock divided by 8<br>110 1/20 of the eTSEC system clock divided by 8<br>111 1/28 of the eTSEC system clock divided by 8 |

### 14.5.3.5.7 MII Management Command Register (MIIMCOM)

The MIIMCOM register is written by the user. Figure 14-45 describes the definition for MIIMCOM.



Offset eTSEC1:0x2_4524 Access: Read/Write

| | 0–29 | 30 | 31 |
|---|---|---|---|
| R<br>W | — | Scan Cycle | Read Cycle |
| Reset | All zeros | | |

**Figure 14-45. MIIMCOM Register Definition**

Table 14-49 describes the fields of the MIIMCOM register.

**Table 14-49. MIIMCOM Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–29 | — | Reserved |

**Table 14-49. MIIMCOM Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 30 | Scan Cycle | Scan cycle. This bit is cleared by default.<br>0 Normal operation.<br>1 The MII management continuously performs read cycles. This is useful for monitoring link fail, for example. |
| 31 | Read Cycle | Read cycle. This bit is cleared by default but is not self-clearing once set.<br>0 Normal operation.<br>1 The MII management performs a single read cycle upon the transition of this bit from 0 to 1 using the PHY address (at MIIMADD[PHY Address]) and the register address (at MIIMADD[Register Address]). The 0-to-1 transition of this bit also causes the MIIMIND[Busy] bit to be set. The read is complete when the MIIMIND[Busy] bit clears. Data is returned in register MIIMSTAT[PHY Status]. |

### 14.5.3.5.8 MII Management Address Register (MIIMADD)

The MIIMADD register is written by the user. Figure 14-46 shows the MIIMADD register.

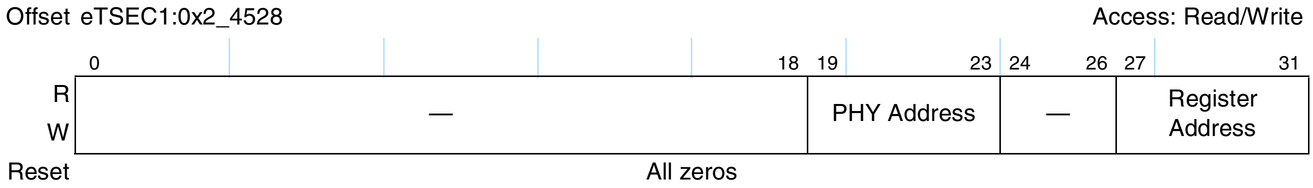


**Figure 14-46. MIIMADD Register Definition**

Table 14-50 describes the fields of the MIIMADD register.

**Table 14-50. MIIMADD Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–18 | — | Reserved |
| 19–23 | PHY Address | This field represents the 5-bit PHY address field of Mgmt cycles. Up to 31 PHYs can be addressed (0 is reserved). Its default value is 0x00. |
| 24–26 | — | Reserved |
| 27–31 | Register Address | This field represents the 5-bit register address field of Mgmt cycles. Up to 32 registers can be accessed. Its default value is 0x00. |

### 14.5.3.5.9 MII Management Control Register (MIIMCON)

MIIMCON, shown in Figure 14-47, is written by the user.

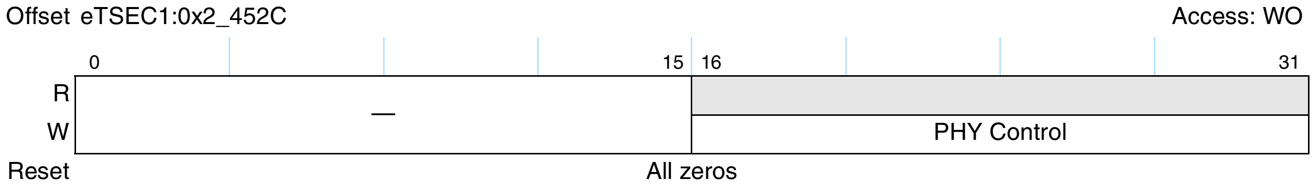


**Figure 14-47. MII Mgmt Control Register Definition**

Table 14-51 describes the fields of the MIIMCON register.

Table 14-51. MIIMCON Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–15 | — | Reserved |
| 16–31 | PHY Control | If written, an MII Mgmt write cycle is performed using this 16-bit data, the pre-configured PHY address (at MIIMADD[PHY Address]) and the register address (at MIIMADD[Register Address]). Its default value is 0x0000. |

### 14.5.3.5.10 MII Management Status Register (MIIMSTAT)

The MIIMSTAT register is read only by the user. Figure 14-48 describes the definition for the MIIMSTAT register.

Offset eTSEC1:0x2_4530 Access: Read only

| | 0 15 | 16 31 |
|---|---|---|
| R | — | PHY Status |
| W | | |
| Reset | All zeros | |

Figure 14-48. MIIMSTAT Register Definition

Table 14-52 describes the fields of the MIIMSTAT register.

Table 14-52. MIIMSTAT Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–15 | — | Reserved |
| 16–31 | PHY Status | Following an MII Mgmt read cycle, the 16-bit data can be read from this location. Its default value is 0x0000. |

### 14.5.3.5.11 MII Management Indicator Register (MIIMIND)

The MIIMIND register is read-only by the user. Figure 14-49 describes the definition for the MIIMIND register.

Offset eTSEC1:0x2_4534 Access: Read only

| | 0 28 | 29 | 30 | 31 |
|---|---|---|---|---|
| R | — | Not Valid | Scan | Busy |
| W | | | | |
| Reset | All zeros | | | |

Figure 14-49. MII Mgmt Indicator Register Definition

Table 14-53. MIIMIND Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–28 | — | Reserved |
| 29 | Not Valid | Not valid.<br>0 MII Mgmt read cycle has completed and the read data is valid.<br>1 MII Mgmt read cycle has not completed and the read data is not yet valid. |
| 30 | Scan | Scan in progress.<br>0 A scan operation (continuous MII Mgmt read cycles) is not in progress.<br>1 A scan operation (continuous MII Mgmt read cycles) is in progress. |
| 31 | Busy | Busy.<br>0 MII Mgmt block is not currently performing an MII Mgmt read or write cycle.<br>1 MII Mgmt block is currently performing an MII Mgmt read or write cycle. |

### 14.5.3.5.12 Interface Status Register (IFSTAT)

Figure 14-50 shows the IFSTAT register.

Offset eTSEC1:0x2_453C;
eTSEC3:0x2_653C

Access: Read only

| | 0 … 21 | 22 | 23 … 31 |
|---|---|---|---|
| R | — | Excess Defer | — |
| W | | | |
| Reset | All zeros | | |

Figure 14-50. Interface Status Register Definition

Table 14-54 describes the fields of the FSTAT register.

Table 14-54. IFSTAT Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–21 | — | Reserved |
| 22 | Excess Defer | Excessive transmission defer. This bit latches high and is cleared when read. This bit is cleared by default.<br>0 Normal operation.<br>1 The MAC excessively defers a transmission. |
| 23–31 | — | Reserved |

### 14.5.3.5.13 MAC Station Address Part 1 Register (MACSTNADDR1)

The MACSTNADDR1 register is written by the user. The value of the station address written into MACSTNADDR1 and MACSTNADDR2 is byte reversed from how it would appear in the DA field of a frame in memory. For example, for a station address of 0x12345678ABCD, MACSTNADDR1 is set to 0xCDAB7856 and MACSTNADDR2 is set to 0x34120000.

Figure 14-51 shows the MACSTNADDR1 register.

Offset eTSEC1:0x2_4540;
eTSEC3:0x2_6540

Access: Read/Write

| | 0 — 7 | 8 — 15 | 16 — 23 | 24 — 31 |
|---|---|---|---|---|
| R<br>W | Station Address, 6th Octet | Station Address, 5th Octet | Station Address, 4th Octet | Station Address, 3rd Octet |
| Reset | All zeros | | | |

**Figure 14-51. MAC Station Address Part 1 Register Definition**

Table 14-55 describes the fields of the MACSTNADDR1 register.

**Table 14-55. MACSTNADDR1 Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 0–7 | Station Address, 6th Octet | This field holds the sixth octet of the station address. The sixth octet (station address bits 40—47) defaults to a value of 0x0. |
| 8–15 | Station Address, 5th Octet | This field holds the fifth octet of the station address. The fifth octet (station address bits 32—39) defaults to a value of 0x0. |
| 16–23 | Station Address, 4th Octet | This field holds the fourth octet of the station address. The fourth octet (station address bits 24—31) defaults to a value of 0x0. |
| 24–31 | Station Address, 3rd Octet | This field holds the third octet of the station address. The third octet (station address bits 16—23) defaults to a value of 0x0. |

### 14.5.3.5.14 MAC Station Address Part 2 Register (MACSTNADDR2)

The MACSTNADDR2 register is written by the user. Figure 14-52 describes the definition for the MACSTNADDR2 register.

Offset eTSEC1:0x2_4544;
eTSEC3:0x2_6544

Access: Read/Write

| | 0 — 7 | 8 — 15 | 16 — 31 |
|---|---|---|---|
| R<br>W | Station Address, 2nd Octet | Station Address, 1st Octet | — |
| Reset | All zeros | | |

**Figure 14-52. MAC Station Address Part 2 Register Definition**

Table 14-56 describes the fields of the MACSTNADDR2 register.

**Table 14-56. MACSTNADDR2 Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 0–7 | Station Address, 2nd Octet | This field holds the second octet of the station address. The second octet (station address bits 8—15) defaults to a value of 0x0. |
| 8–15 | Station Address, 1st Octet | This field holds the first octet of the station address. The first octet (station address bits 0—7) defaults to a value of 0x0. |
| 16–31 | — | Reserved |

### 14.5.3.5.15 MAC Exact Match Address 1–15 Part 1 Registers (MAC01ADDR1–MAC15ADDR1)

The MAC01ADDR1–MAC15ADDR1 registers are written by the user with the unicast or multicast addresses aliasing the MAC. Figure 14-53 describes the definition for all of the fifteen MAC*n*ADDR1 registers. The value of the address written into MAC*x*ADDR1 and MAC*n*ADDR2 is byte reversed from how it would appear in the DA field of a frame in memory. For example, for a MAC address of 0x12345678ABCD, MACnADDR1 is set to 0xCDAB7856 and MACnADDR2 is set to 0x34120000. For any valid, non-zero MAC address received, exact match registers can be excluded individually by clearing them to all zero bytes.

Offset eTSEC1:0x2_4548+8×*n*;
eTSEC3:0x2_6548+8×*n*

Access: Read/Write

| | 0 7 | 8 15 | 16 23 | 24 31 |
|---|---|---|---|---|
| R<br>W | Exact Match Address, 6th Octet | Exact Match Address, 5th Octet | Exact Match Address, 4th Octet | Exact Match Address, 3rd Octet |
| Reset | All zeros | | | |

**Figure 14-53. MAC Exact Match Address *n* Part 1 Register Definition**

Table 14-55 describes the fields of a MAC*n*ADDR1 register.

**Table 14-57. MAC*n*ADDR1 Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 0–7 | Exact Match Address, 6th Octet | Holds the sixth octet of the exact match address. The sixth octet (destination address bits 40—47) defaults to a value of 0x0. |
| 8–15 | Exact Match Address, 5th Octet | Holds the fifth octet of the exact match address. The fifth octet (destination address bits 32—39) defaults to a value of 0x0. |
| 16–23 | Exact Match Address, 4th Octet | Holds the fourth octet of the exact match address. The fourth octet (destination address bits 24—31) defaults to a value of 0x0. |
| 24–31 | Exact Match Address, 3rd Octet | Holds the third octet of the exact match address. The third octet (destination address bits 16—23) defaults to a value of 0x0. |

### 14.5.3.5.16 MAC Exact Match Address 1–15 Part 2 Registers (MAC01ADDR2–MAC15ADDR2)

The MAC01ADDR2–MAC15ADDR2 registers are written by the user with the unicast or multicast addresses aliasing the MAC. Figure 14-54 describes the definition for all of the fifteen MACxADDR2 registers.

Offset eTSEC1:0x2_454C+8×*n*;
eTSEC3:0x2_654C+8×*n*

Access: Read/Write

| | 0 7 | 8 15 | 16 31 |
|---|---|---|---|
| R<br>W | Exact Match Address, 2nd Octet | Exact Match Address, 1st Octet | — |
| Reset | All zeros | | |

**Figure 14-54. MAC Exact Match Address *x* Part 2 Register Definition**

Table 14-56 describes the fields of a MAC*x*ADDR2 register.

**Table 14-58. MAC01ADDR2–MAC15ADDR2 Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 0–7 | Exact Match Address, 2nd Octet | This field holds the second octet of the exact match address. The second octet (destination address bits 8–15) defaults to a value of 0x0. |
| 8–15 | Exact Match Address, 1st Octet | This field holds the first octet of the exact match address. The first octet (destination address bits 0–7) defaults to a value of 0x0. |
| 16–31 | — | Reserved |

### 14.5.3.6 MIB Registers

This section describes the MIB registers. The eTSEC RMON module has 37 separate statistics counters, which simply count or accumulate statistical events that occur as packets transmitted and received. These counters support RMON MIB group 1, RMON MIB group 2 if table counters, RMON MIB group 3, RMON MIB group 9, RMON MIB 2, and the IEEE 802.3 Ethernet MIB.

An interrupt can be generated upon any one counter’s rollover condition through a carry interrupt output from the RMON. Each counter’s rollover condition can be discretely masked from causing an interrupt by internal masking registers. In addition, each individual counter value may be reset on read access, or all counters may be simultaneously reset by setting ECNTRL[CLRCNT].

The majority of MIB counters are Ethernet-specific.

In FIFO modes, only the following registers are updated:

- Transmit: TBYT, TPKT, TDRP
- Receive: RBYT, RPKT, RFCS

**NOTE**

RMON counters do not comprehend custom VLAN tagged frames. Affected counters include TRMGV, RMCA, RBCA, RXCF, RXPF, RXUO, RALN, RFLR, ROVR, RJBR, TMCA, TBCA, TXPF, TXCF. Specifically, custom VLAN tagged frames are not afforded the ability to be greater than 1518, as compared to the IEEE standard tagged frames.

**NOTE**

The transmit and receive frame counters (TR64, TR127, TR 255, TR511, TR1K, TRMAX, adn TRMGV) do not increment for aborted frames (collision retry limit exceeded, late collision, underrurn, EBERR, TxFIFO data error, frame truncated due to exceeding MAXFRM, or excessive deferral).

#### 14.5.3.6.1 Transmit and Receive 64-Byte Frame Counter (TR64)

Figure 14-55 describes the definition for the TR64 register.

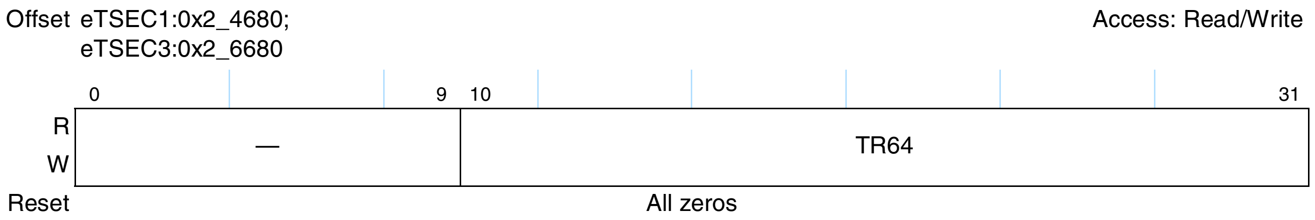


**Figure 14-55. Transmit and Receive 64-Byte Frame Register Definition**

Table 14-59 describes the fields of the TR64 register.

**Table 14-59. TR64 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–9 | — | Reserved |
| 10–31 | TR64 | Transmit and receive 64-byte frame counter—Increment for each good or bad frame transmitted and received which is 64 bytes in length, inclusive (excluding preamble and SFD but including FCS bytes). |

#### 14.5.3.6.2 Transmit and Receive 65- to 127-Byte Frame Counter (TR127)

Figure 14-56 describes the definition for the TR127 register.

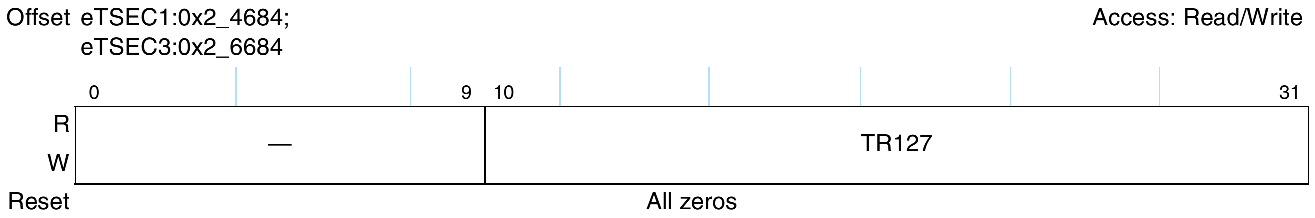


**Figure 14-56. Transmit and Receive 65- to 127-Byte Frame Register Definition**

Table 14-60 describes the fields of the TR127 register.

**Table 14-60. TR127 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–9 | — | Reserved |
| 10–31 | TR127 | Transmit and receive 65- to 127-byte frame counter—Increments for each good or bad frame transmitted and received which is 65–127 bytes in length, inclusive (excluding preamble and SFD but including FCS bytes). |

### 14.5.3.6.3 Transmit and Receive 128- to 255-Byte Frame Counter (TR255)

Figure 14-57 describes the definition for the TR255 register.

Offset eTSEC1:0x2_4688;
eTSEC3:0x2_6688

Access: Read/Write

| | 0–9 | 10–31 |
|---|---|---|
| R<br>W | — | TR255 |
| Reset | All zeros | |

**Figure 14-57. Transmit and Received 128- to 255-Byte Frame Register Definition**

Table 14-61 describes the fields of the TR255 register.

**Table 14-61. TR255 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–9 | — | Reserved |
| 10–31 | TR255 | Transmit and receive 128- to 255-byte frame counter—Increments for each good or bad frame transmitted and received which is 128–255 bytes in length, inclusive (excluding preamble and SFD but including FCS bytes). |

### 14.5.3.6.4 Transmit and Receive 256- to 511-Byte Frame Counter (TR511)

Figure 14-58 describes the definition for the TR511 register.

Offset eTSEC1:0x2_468C;
eTSEC3:0x2_668C

Access: Read/Write

| | 0–9 | 10–31 |
|---|---|---|
| R<br>W | — | TR511 |
| Reset | All zeros | |

**Figure 14-58. Transmit and Received 256- to 511-Byte Frame Register Definition**

Table 14-62 describes the fields of the TR511 register.

**Table 14-62. TR511 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–9 | — | Reserved |
| 10–31 | TR511 | Increments for each good or bad frame transmitted and received which is 256–511 bytes in length, inclusive (excluding preamble and SFD but including FCS bytes). |

### 14.5.3.6.5 Transmit and Receive 512- to 1023-Byte Frame Counter (TR1K)

Figure 14-59 shows the TR1K register.

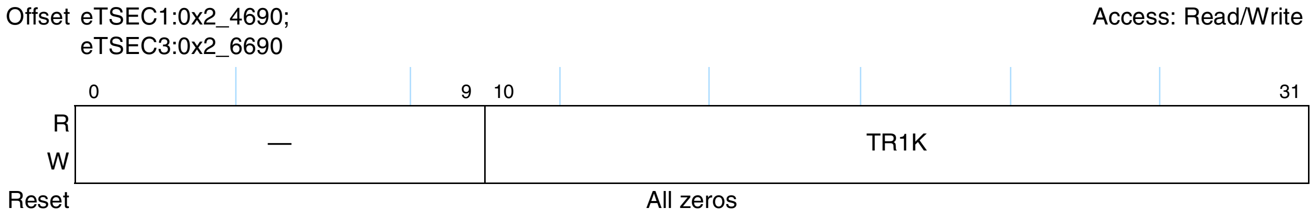


**Figure 14-59. Transmit and Received 512- to 1023-Byte Frame Register Definition**

Table 14-63 describes the fields of the TR1K register.

**Table 14-63. TR1K Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–9 | — | Reserved |
| 10–31 | TR1K | Increments for each good or bad frame transmitted and received which is 512–1023 bytes in length, inclusive (excluding preamble and SFD but including FCS bytes). |

### 14.5.3.6.6 Transmit and Receive 1024- to 1518-Byte Frame Counter (TRMAX)

Figure 14-60 describes the definition for the TRMAX register.

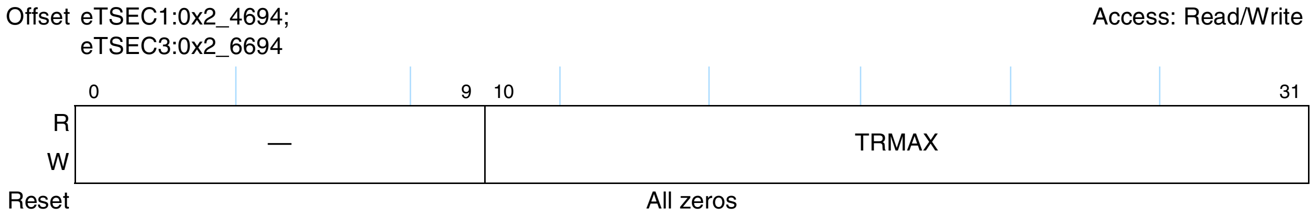


**Figure 14-60. Transmit and Received 1024- to 1518-Byte Frame Register Definition**

Table 14-64 describes the fields of the TRMAX register.

**Table 14-64. TRMAX Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–9 | — | Reserved |
| 10–31 | TRMAX | Increments for each good or bad frame transmitted and received which is 1024–1518 bytes in length, inclusive (excluding preamble and SFD but including FCS bytes). |

#### 14.5.3.6.7 Transmit and Receive 1519- to 1522-Byte VLAN Frame Counter (TRMGV)

Figure 14-61 describes the definition for the TRMGV register.

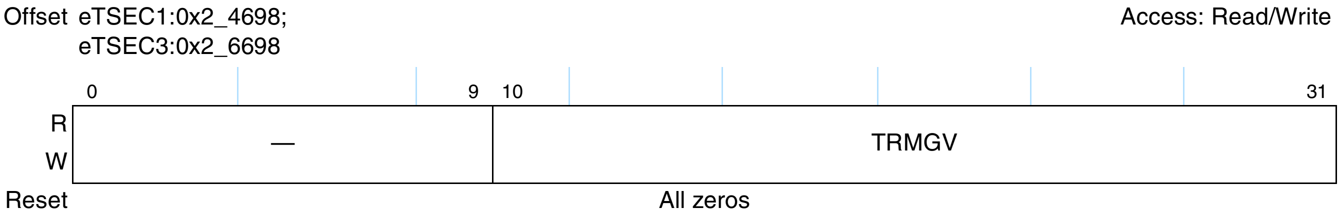


**Figure 14-61. Transmit and Received 1519- to 1522-Byte VLAN Frame Register Definition**

Table 14-65 describes the fields of the TRMGV register.

**Table 14-65. TRMGV Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–9 | — | Reserved |
| 10–31 | TRMGV | Increments for each good or bad frame transmitted and received which is 1519–1522 bytes in length, inclusive (excluding preamble and SFD but including FCS bytes). |

#### 14.5.3.6.8 Receive Byte Counter (RBYT)

Figure 14-62 shows the RBYT register.

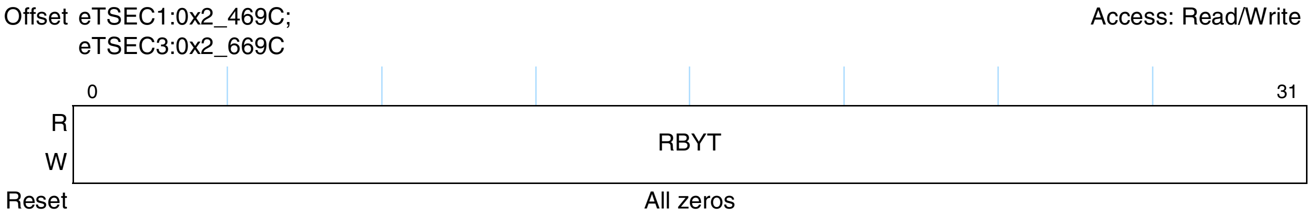


**Figure 14-62. Receive Byte Counter Register Definition**

Table 14-66 describes the fields of the RBYT register.

**Table 14-66. RBYT Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–31 | RBYT | Receive byte counter. The statistic counter register increments by the byte count of frames received, including those in bad packets, excluding preamble and SFD but including FCS bytes. In FIFO mode, all bytes (including FCS bytes) are counted. |

### 14.5.3.6.9 Receive Packet Counter (RPKT)

Figure 14-63 describes the definition for the RPKT register.

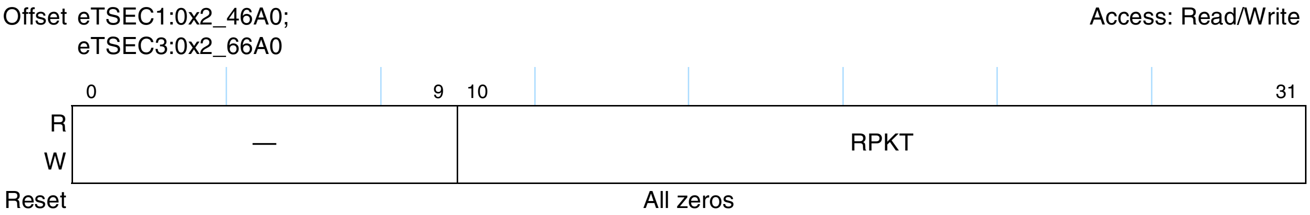


**Figure 14-63. Receive Packet Counter Register Definition**

Table 14-67 describes the fields of the RPKT register.

**Table 14-67. RPKT Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–9 | — | Reserved |
| 10-31 | RPKT | Receive packet counter. Increments for each frame received packet (including bad packets, all unicast, broadcast, and multicast packets). |

### 14.5.3.6.10 Receive FCS Error Counter (RFCS)

Figure 14-64 describes the definition for the RFCS register.

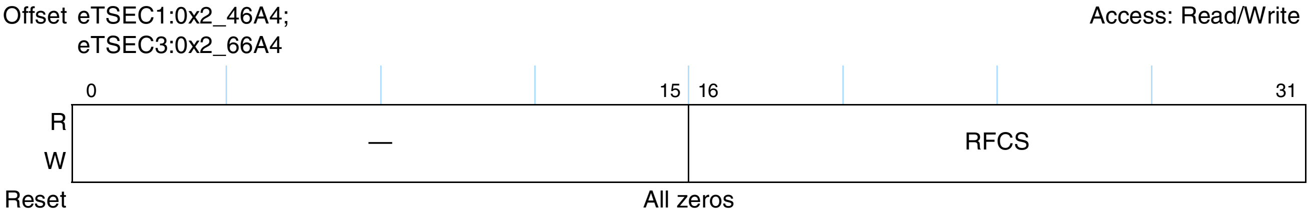


**Figure 14-64. Receive FCS Error Counter Register Definition**

Table 14-68 describes the fields of the RFCS register.

**Table 14-68. RFCS Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–15 | — | Reserved |
| 16–31 | RFCS | Receive FCS error counter. In Ethernet mode, increments for each frame received that has an integral 64–1518 length and contains a frame check sequence error. In FIFO mode, increments for each frame received that contains a frame check sequence error (regardless of size). |

### 14.5.3.6.11 Receive Multicast Packet Counter (RMCA)

Figure 14-65 describes the definition for the RMCA register.

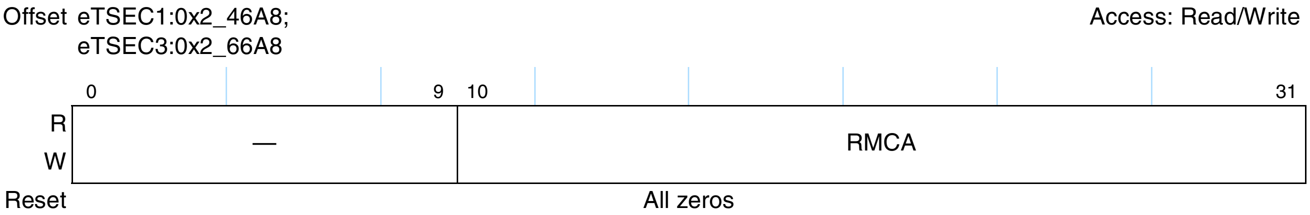


**Figure 14-65. Receive Multicast Packet Counter Register Definition**

Table 14-69 describes the fields of the RMCA register.

**Table 14-69. RMCA Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–9 | — | Reserved |
| 10–31 | RMCA | Receive multicast packet counter. Increments for each multicast frame with valid CRC and of lengths 64 to 1518 (non VLAN) or 1522 (VLAN), excluding broadcast frames. This count does not include range/length errors. |

### 14.5.3.6.12 Receive Broadcast Packet Counter (RBCA)

Figure 14-66 describes the definition for the RBCA register.

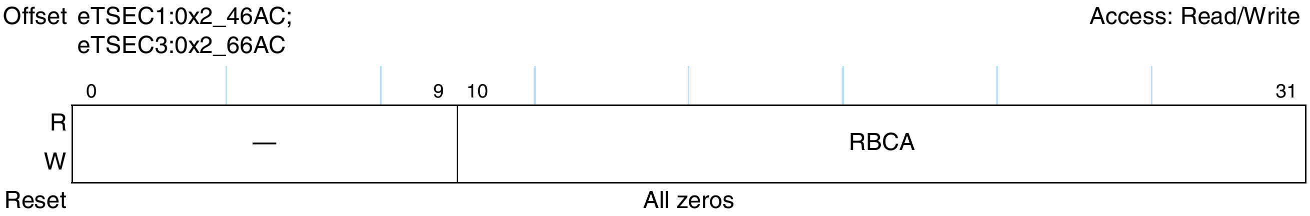


**Figure 14-66. Receive Broadcast Packet Counter Register Definition**

Table 14-70 describes the fields of the RBCA register.

**Table 14-70. RBCA Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–9 | — | Reserved |
| 10–31 | RBCA | Receive broadcast packet counter. Increments for each broadcast frame with valid CRC and of lengths 64 to 1518 (non VLAN) or 1522 (VLAN), excluding multicast frames. Does not include range/length errors. |

### 14.5.3.6.13 Receive Control Frame Packet Counter (RXCF)

Figure 14-67 describes the definition for the RXCF register.

Offset eTSEC1:0x2_46B0;
eTSEC3:0x2_66B0

Access: Read/Write

|  | 0 — 15 | 16 — 31 |
|---|---|---|
| R<br>W | — | RXCF |
| Reset | All zeros | |

Figure 14-67. Receive Control Frame Packet Counter Register Definition

Table 14-71 describes the fields of the RXCF register.

Table 14-71. RXCF Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–15 | — | Reserved |
| 16–31 | RXCF | Receive control frame packet counter. Increments for each MAC control frame received (PAUSE and unsupported) with valid CRC and of lengths 64 to 1518 (non VLAN) or 1522 (VLAN). |

### 14.5.3.6.14 Receive Pause Frame Packet Counter (RXPF)

Figure 14-68 describes the definition for the RXPF register.

Offset eTSEC1:0x2_46B4;
eTSEC3:0x2_66B4

Access: Read/Write

|  | 0 — 15 | 16 — 31 |
|---|---|---|
| R<br>W | — | RXPF |
| Reset | All zeros | |

Figure 14-68. Receive Pause Frame Packet Counter Register Definition

Table 14-72 describes the fields of the RXPF register.

Table 14-72. RXPF Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–15 | — | Reserved |
| 16–31 | RXPF | Receive PAUSE frame packet counter. Increments each time a PAUSE MAC control frame is received with valid CRC and of lengths 64 to 1518 (non VLAN) or 1522 (VLAN). |

#### 14.5.3.6.15 Receive Unknown Opcode Packet Counter (RXUO)

Figure 14-69 describes the definition for the RXUO register.

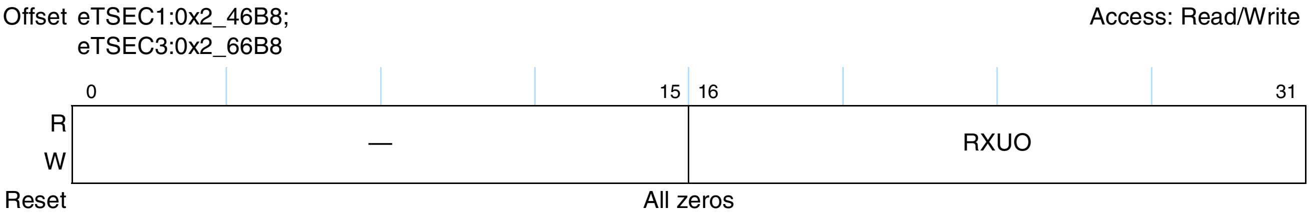


**Figure 14-69. Receive Unknown OPCode Packet Counter Register Definition**

Table 14-73 describes the fields of the RXUO register.

**Table 14-73. RXUO Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–15 | — | Reserved |
| 16–31 | RXUO | Receive unknown opcode counter. Increments each time a MAC control frame is received which contains an opcode other than PAUSE, but the frame has valid CRC and length 64 to 1518 (non VLAN) or 1522 (VLAN). |

#### 14.5.3.6.16 Receive Alignment Error Counter (RALN)

Figure 14-70 describes the definition for the RALN register.

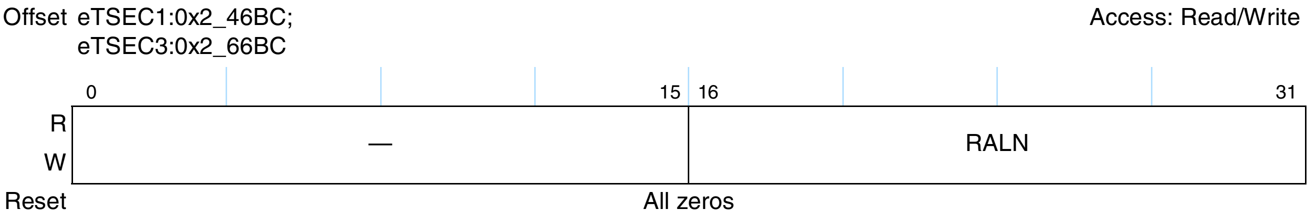


**Figure 14-70. Receive Alignment Error Counter Register Definition**

Table 14-74 describes the fields of the RALN register.

**Table 14-74. RALN Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–15 | — | Reserved |
| 16–31 | RALN | Receive alignment error counter. Increments for each received frame from 64 to 1518 (non VLAN) or 1522 (VLAN) which contains an invalid FCS and is not an integral number of bytes. |

### 14.5.3.6.17 Receive Frame Length Error Counter (RFLR)

Figure 14-71 describes the definition for the RFLR register.

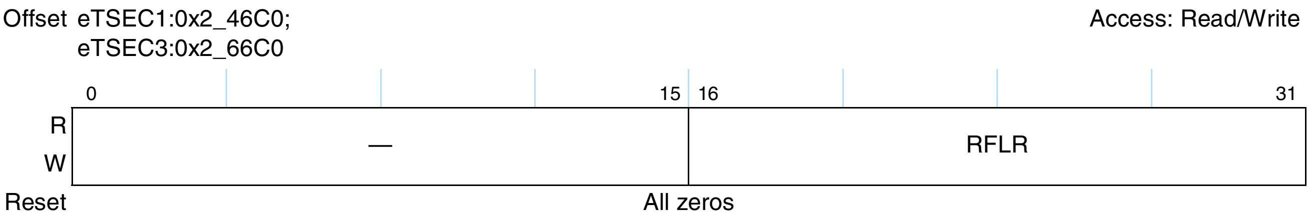


**Figure 14-71. Receive Frame Length Error Counter Register Definition**

Table 14-75 describes the fields of the RFLR register.

**Table 14-75. RFLR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–15 | — | Reserved |
| 16–31 | RFLR | Receive frame length error counter. Increments for each frame received in which the 802.3 length field did not match the number of data bytes actually received (46–1500 bytes). The counter does not increment if the length field is not a valid 802.3 length, such as an Ethertype value. |

### 14.5.3.6.18 Receive Code Error Counter (RCDE)

Figure 14-72 describes the definition for the RCDE register.

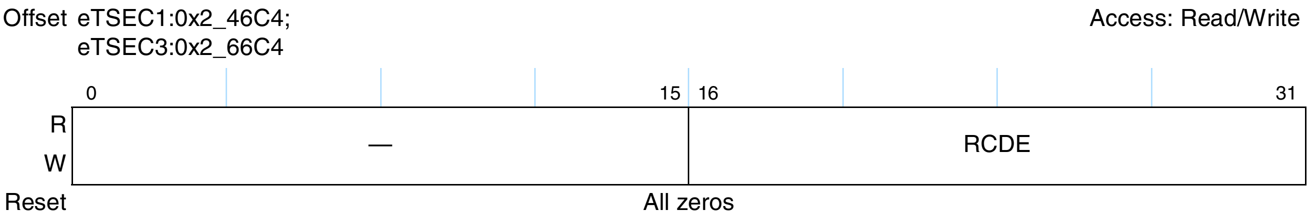


**Figure 14-72. Receive Code Error Counter Register Definition**

Table 14-76 describes the fields of the RCDE register.

**Table 14-76. RCDE Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–15 | — | Reserved |
| 16–31 | RCDE | Receive code error counter. Increments each time a valid carrier is present and at least one invalid data symbol is detected. |

### 14.5.3.6.19 Receive Carrier Sense Error Counter (RCSE)

Figure 14-73 describes the definition for the RCSE register.

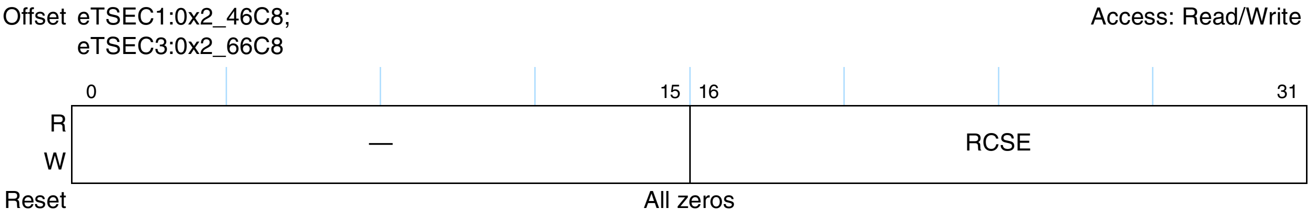


**Figure 14-73. Receive Carrier Sense Error Counter Register Definition**

Table 14-77 describes the fields of the RCSE register.

**Table 14-77. RCSE Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–15 | — | Reserved |
| 16–31 | RCSE | Receive false carrier counter. Counts the number of times that the carrier sense condition was lost or never asserted when attempting to transmit a frame on a particular interface.<br>The count represented by an instance of this object is incremented at most once per transmission attempt, even if the carrier sense condition fluctuates during a transmission attempt. The event is reported along with the statistics generated on the next received frame, as defined by a 1 on TSEC*n*_RX_ER and an 0xE on TSEC*n*_RXD. Only one false carrier condition can be detected and logged between frames. |

### 14.5.3.6.20 Receive Undersize Packet Counter (RUND)

Figure 14-74 describes the definition for the RUND register.

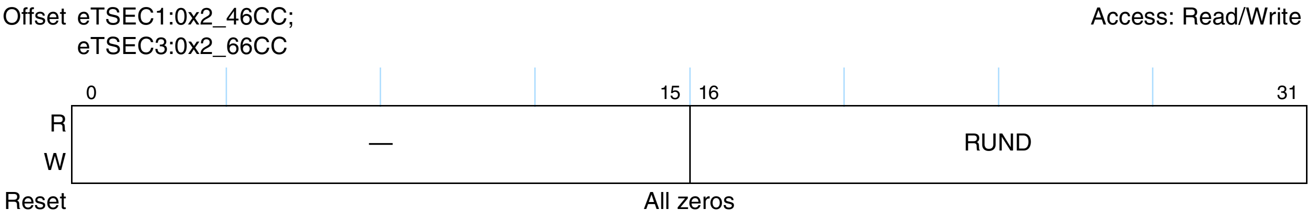


**Figure 14-74. Receive Undersize Packet Counter Register Definition**

Table 14-78 describes the fields of the RUND register.

**Table 14-78. RUND Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–15 | — | Reserved |
| 16–31 | RUND | Receive undersize packet counter. Increments each time a frame is received which is less than 64 bytes in length and contains a valid FCS and were otherwise well formed. This count does not include range length errors. |

### 14.5.3.6.21 Receive Oversize Packet Counter (ROVR)

Figure 14-75 describes the definition for the ROVR register.

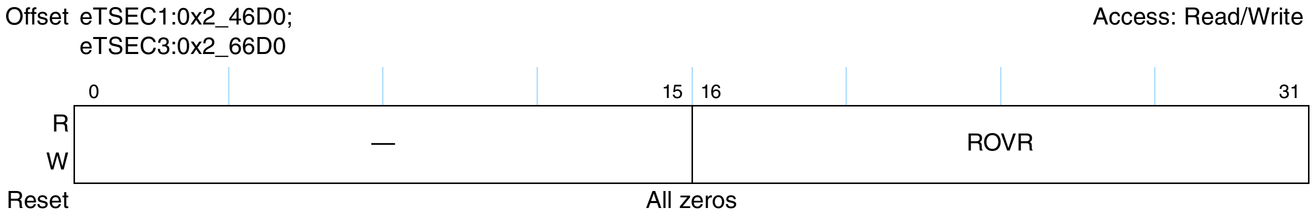


**Figure 14-75. Receive Oversize Packet Counter Register Definition**

Table 14-79 describes the fields of the ROVR register.

**Table 14-79. ROVR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–15 | — | Reserved |
| 16–31 | ROVR | Receive oversize packet counter. Increments each time a frame is received which exceeded 1518 (non VLAN) or 1522 (VLAN) and contains a valid FCS and was otherwise well formed. |

### 14.5.3.6.22 Receive Fragments Counter (RFRG)

Figure 14-76 describes the definition for the RFRG register.

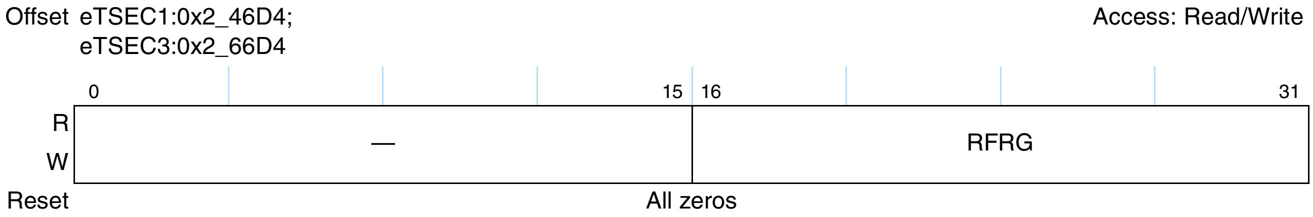


**Figure 14-76. Receive Fragments Counter Register Definition**

Table 14-80 describes the fields of the RFRG register.

**Table 14-80. RFRG Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–15 | — | Reserved |
| 16–31 | RFRG | Receive fragments counter. Increments for each frame received which is less than 64 bytes in length and contains an invalid FCS. This includes integral and non-integral lengths. |

### 14.5.3.6.23 Receive Jabber Counter (RJBR)

Figure 14-77 describes the definition for the RJBR register.

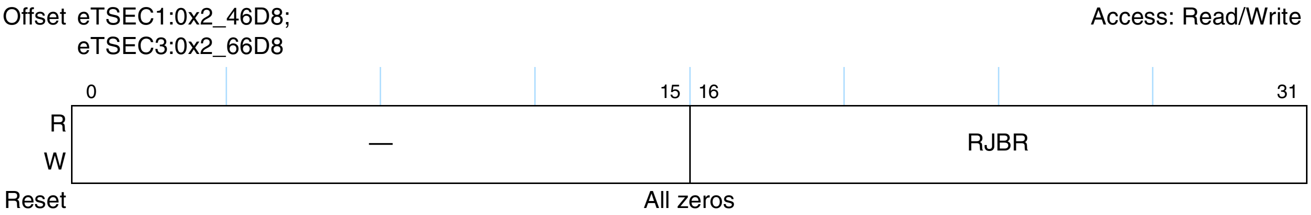


**Figure 14-77. Receive Jabber Counter Register Definition**

Table 14-81 describes the fields of the RJBR register.

**Table 14-81. RJBR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–15 | — | Reserved |
| 16–31 | RJBR | Receive jabber counter. Increments for frames received which exceed 1518 (non VLAN) or 1522 (VLAN) bytes and contain an invalid FCS. This includes alignment errors. |

### 14.5.3.6.24 Receive Dropped Packet Counter (RDRP)

Figure 14-78 describes the definition for the RDRP register.

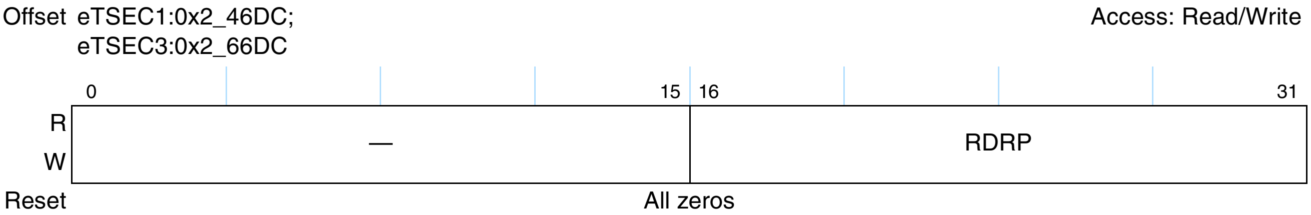


**Figure 14-78. Receive Dropped Packet Counter Register Definition**

Table 14-82 describes the fields of the RDRP register.

**Table 14-82. RDRP Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–15 | — | Reserved |
| 16–31 | RDRP | Receive dropped packets counter. Increments for frames received which are streamed to system but are later dropped due to lack of system resources. |

### 14.5.3.6.25 Transmit Byte Counter (TBYT)

Figure 14-79 depicts the TBYT register.

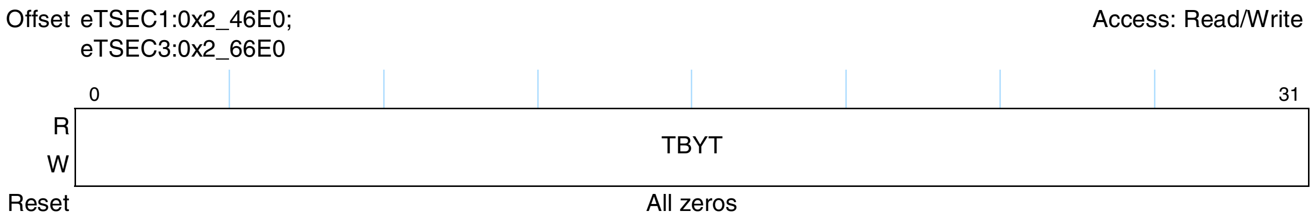


**Figure 14-79. Transmit Byte Counter Register Definition**

Table 14-83 describes the fields of the TBYT register.

**Table 14-83. TBYT Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–31 | TBYT | Transmit byte counter. Increments by the number of bytes that were put on the wire including fragments of frames that were involved with collisions. This count does not include preamble/SFD or jam bytes, except for half-duplex flow control (back-pressure triggered by TCTRL[THDF]=1). For THDF, the sum total of ‘phantom’ preamble bytes transmitted for flow control purposes is included in the TBYT increment value of the next frame to be transmitted, up to 65,535 bytes of frame and phantom preamble.<br>Note that the value of TBYT may be greater than the actual number of bytes transmitted if the frame is truncated because it exceeds MAXFRM. |

### 14.5.3.6.26 Transmit Packet Counter (TPKT)

Figure 14-80 describes the definition for the TPKT register.

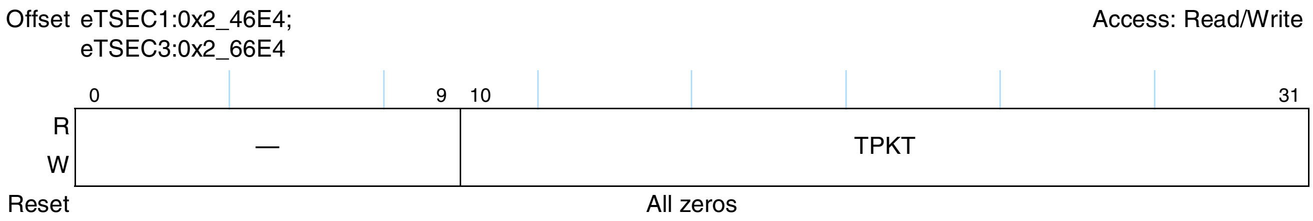


**Figure 14-80. Transmit Packet Counter Register Definition**

Table 14-84 describes the fields of the TPKT register.

**Table 14-84. TPKT Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–9 | — | Reserved |
| 10–31 | TPKT | Transmit packet counter. Increments for each transmitted packet (including bad packets, excessive deferred packets, excessive collision packets, late collision packets, all unicast, broadcast, and multicast packets). |

### 14.5.3.6.27 Transmit Multicast Packet Counter (TMCA)

Figure 14-81 describes the definition for the TMCA register.

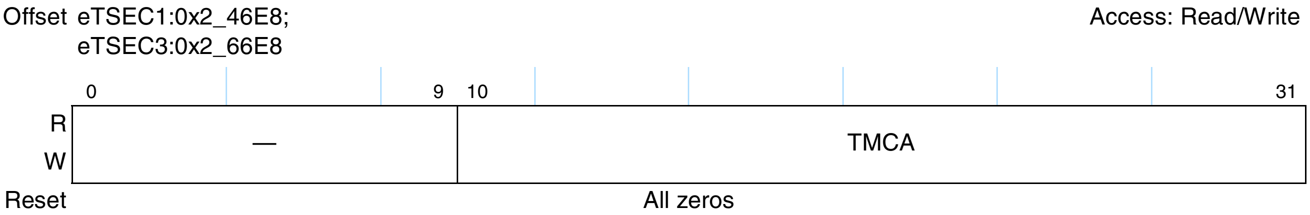


**Figure 14-81. Transmit Multicast Packet Counter Register Definition**

Table 14-85 describes the fields of the TMCA register.

**Table 14-85. TMCA Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–9 | — | Reserved |
| 10–31 | TMCA | Transmit multicast packet counter. Increments for each multicast valid frame transmitted (excluding broadcast frames) with valid CRC and of lengths 64 to 1518 (non VLAN) or 1522 (VLAN). |

### 14.5.3.6.28 Transmit Broadcast Packet Counter (TBCA)

Figure 14-82 describes the definition for the TBCA register.

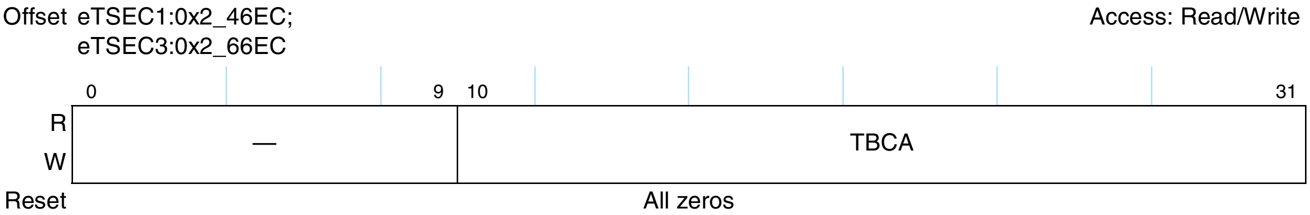


**Figure 14-82. Transmit Broadcast Packet Counter Register Definition**

Table 14-86 describes the fields of the TBCA register.

**Table 14-86. TBCA Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–9 | — | Reserved |
| 10–31 | TBCA | Transmit broadcast packet counter. Increments for each broadcast frame transmitted (excluding multicast frames) with valid CRC and of lengths 64 to 1518 (non VLAN) or 1522 (VLAN). |

### 14.5.3.6.29 Transmit Pause Control Frame Counter (TXPF)

Figure 14-83 describes the definition for the TXPF register.

Offset eTSEC1:0x2_46F0;
eTSEC3:0x2_66F0

Access: Read/Write

| | 0–15 | 16–31 |
|---|---|---|
| R<br>W | — | TXPF |
| Reset | All zeros | |

**Figure 14-83. Transmit Pause Control Frame Counter Register Definition**

Table 14-87 describes the fields of the TXPF register.

**Table 14-87. TXPF Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–15 | — | Reserved |
| 16–31 | TXPF | Transmit PAUSE frame packet counter. Increments each time a valid PAUSE MAC control frame is transmitted with valid CRC and of lengths 64 to 1518 (non VLAN) or 1522 (VLAN). |

### 14.5.3.6.30 Transmit Deferral Packet Counter (TDFR)

Figure 14-84 describes the definition for the TDFR register.

Offset eTSEC1:0x2_46F4;
eTSEC3:0x2_66F4

Access: Read/Write

| | 0–19 | 20–31 |
|---|---|---|
| R<br>W | — | TDFR |
| Reset | All zeros | |

**Figure 14-84. Transmit Deferral Packet Counter Register Definition**

Table 14-88 describes the fields of the TDFR register.

**Table 14-88. TDFR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–19 | — | Reserved |
| 20–31 | TDFR | Transmit deferral packet counter. Increments for each frame, which was deferred on its first transmission attempt. This count does not include frames involved in collisions. |

### 14.5.3.6.31 Transmit Excessive Deferral Packet Counter (TEDF)

Figure 14-85 describes the definition for the TEDF register.

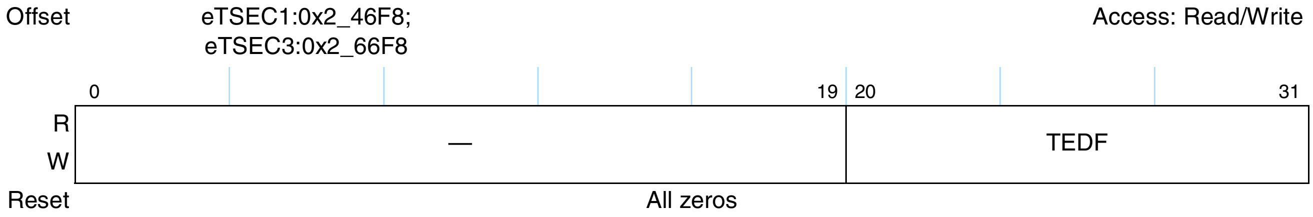


**Figure 14-85. Transmit Excessive Deferral Packet Counter Register Definition**

Table 14-89 describes the fields of the TEDF register.

**Table 14-89. TEDF Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–19 | — | Reserved |
| 20–31 | TEDF | Transmit excessive deferral packet counter. Increments for frames aborted which were deferred for an excessive period of time (3036 byte times). |

### 14.5.3.6.32 Transmit Single Collision Packet Counter (TSCL)

Figure 14-86 describes the definition for the TSCL register.

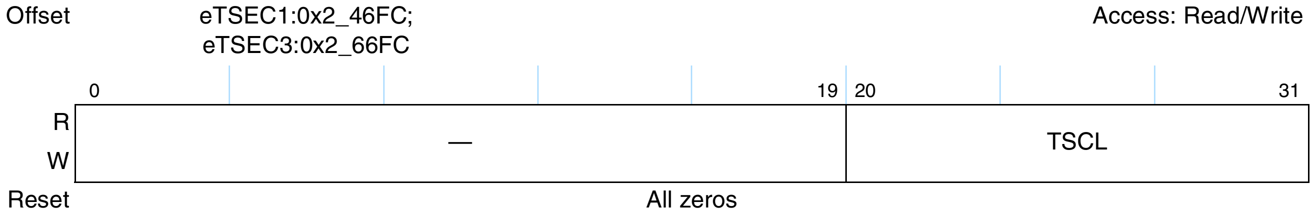


**Figure 14-86. Transmit Single Collision Packet Counter Register Definition**

Table 14-90 describes the fields of the TSCL register.

**Table 14-90. TSCL Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–19 | — | Reserved |
| 20–31 | TSCL | Transmit single collision packet counter. Increments for each frame transmitted which experienced exactly one collision during transmission. |

### 14.5.3.6.33 Transmit Multiple Collision Packet Counter (TMCL)

Figure 14-87 describes the definition for the TMCL register.

Offset eTSEC1:0x2_4700;
eTSEC3:0x2_6700

Access: Read/Write

|  | 0–19 | 20–31 |
|---|---|---|
| R<br>W | — | TMCL |
| Reset | All zeros | |

**Figure 14-87. Transmit Multiple Collision Packet Counter Register Definition**

Table 14-91 describes the fields of the TMCL register.

**Table 14-91. TMCL Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–19 | — | Reserved |
| 20–31 | TMCL | Transmit multiple collision packet counter. Increments for each frame transmitted which experienced 2–15 collisions (including any late collisions) during transmission as defined using the Half_Duplex[RETRANSMISSION MAXIMUM] field. |

### 14.5.3.6.34 Transmit Late Collision Packet Counter (TLCL)

Figure 14-88 describes the definition for the TLCL register.

Offset eTSEC1:0x2_4704;
eTSEC3:0x2_6704

Access: Read/Write

|  | 0–19 | 20–31 |
|---|---|---|
| R<br>W | — | TLCL |
| Reset | All zeros | |

**Figure 14-88. Transmit Late Collision Packet Counter Register Definition**

Table 14-92 describes the fields of the TLCL register.

**Table 14-92. TLCL Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–19 | — | Reserved |
| 20–31 | TLCL | Transmit late collision packet counter. Increments for each frame transmitted which experienced a late collision during a transmission attempt. Late collisions are defined using the collision window field of the half-duplex [26:31] register. |

### 14.5.3.6.35 Transmit Excessive Collision Packet Counter (TXCL)

Figure 14-89 describes the definition for the TXCL register.

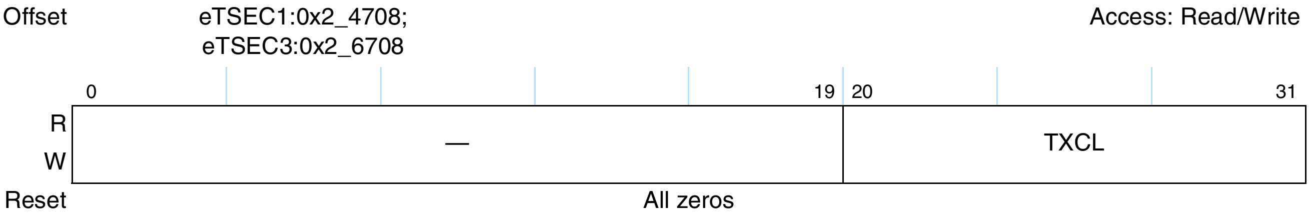


**Figure 14-89. Transmit Excessive Collision Packet Counter Register Definition**

Table 14-93 describes the fields of the TXCL register.

**Table 14-93. TXCL Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–19 | — | Reserved |
| 20–31 | TXCL | Transmit excessive collision packet counter. Increments for each frame that experienced 16 collisions during transmission and was aborted. |

### 14.5.3.6.36 Transmit Total Collision Counter (TNCL)

Figure 14-90 describes the definition for the TNCL register.

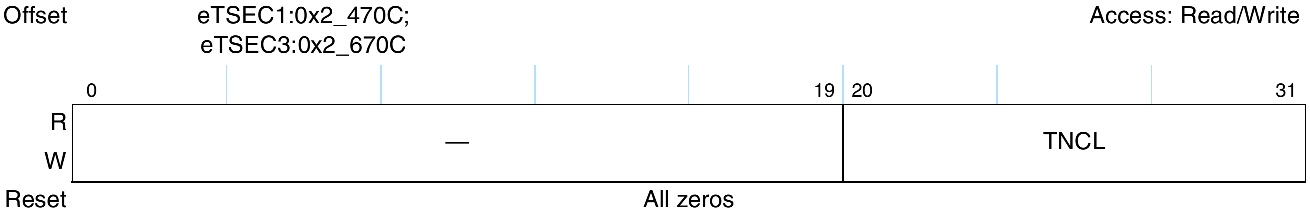


**Figure 14-90. Transmit Total Collision Counter Register Definition**

Table 14-94 describes the fields of the TNCL register.

**Table 14-94. TNCL Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–19 | — | Reserved |
| 20–31 | TNCL | Transmit total collision counter. Increments by the number of collisions experienced during the transmission of a frame as defined as the simultaneous presence of signals on the DO and RD circuits (That is, transmitting and receiving at the same time). **Note:** This count does not include collisions that result in an excessive collision condition. |

### 14.5.3.6.37 Transmit Drop Frame Counter (TDRP)

Figure 14-91 describes the definition for the TDRP register.

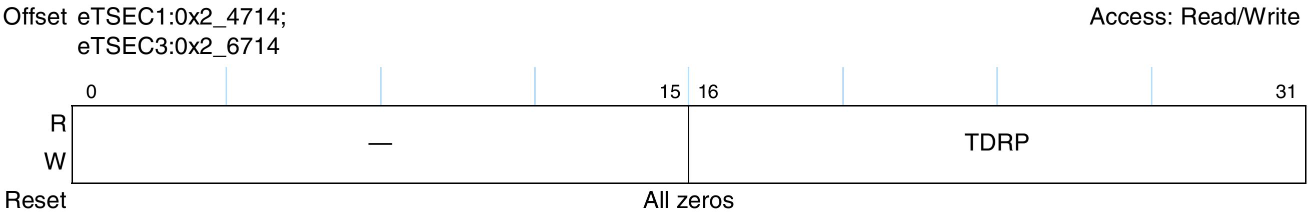


**Figure 14-91. Transmit Drop Frame Counter Register Definition**

Table 14-95 describes the fields of the TDRP register.

**Table 14-95. TDRP Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–15 | — | Reserved |
| 16–31 | TDRP | Transmit drop frame counter. Increments each time a memory error or an underrun has occurred. |

### 14.5.3.6.38 Transmit Jabber Frame Counter (TJBR)

Figure 14-92 describes the definition for the TJBR register.

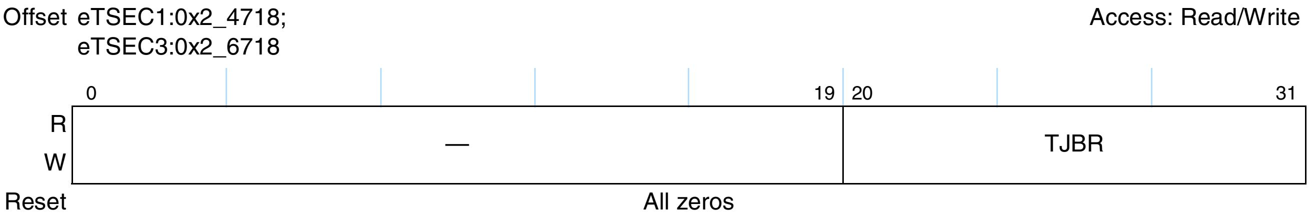


**Figure 14-92. Transmit Jabber Frame Counter Register Definition**

Table 14-96 describes the fields of the TJBR register.

**Table 14-96. TJBR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–19 | — | Reserved |
| 20–31 | TJBR | Transmit jabber frame counter. Increments for each oversized transmitted frame with an incorrect FCS value. |

#### 14.5.3.6.39 Transmit FCS Error Counter (TFCS)

Figure 14-93 describes the definition for the TFCS register.

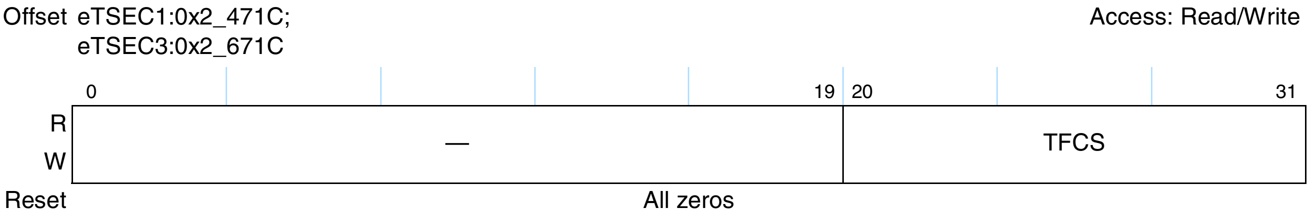


**Figure 14-93. Transmit FCS Error Counter Register Definition**

Table 14-97 describes the fields of the TFCS register.

**Table 14-97. TFCS Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–19 | — | Reserved |
| 20–31 | TFCS | Transmit FCS error counter. Increments for every valid sized packet with an incorrect FCS value. |

#### 14.5.3.6.40 Transmit Control Frame Counter (TXCF)

Figure 14-94 describes the definition for the TXCF register.

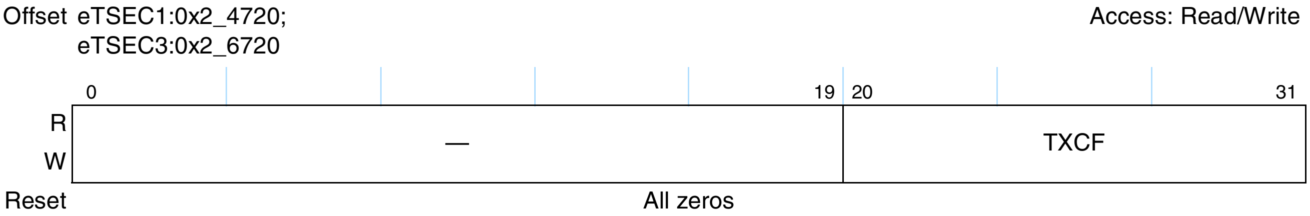


**Figure 14-94. Transmit Control Frame Counter Register Definition**

Table 14-98 describes the fields of the TXCF register.

**Table 14-98. TXCF Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–19 | — | Reserved |
| 20–31 | TXCF | Transmit control frame counter. Increments for every control frame with valid CRC and of lengths 64 to 1518 (non VLAN) or 1522 (VLAN). |

### 14.5.3.6.41 Transmit Oversize Frame Counter (TOVR)

Figure 14-95 describes the definition for the TOVR register.

Offset eTSEC1:0x2_4724;
eTSEC3:0x2_6724

Access: Read/Write

| | 0–19 | 20–31 |
|---|---|---|
| R<br>W | — | TOVR |
| Reset | All zeros | |

**Figure 14-95. Transmit Oversized Frame Counter Register Definition**

Table 14-99 describes the fields of the TOVR register.

**Table 14-99. TOVR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–19 | — | Reserved |
| 20–31 | TOVR | Transmit oversize frame counter. Increments each time a frame is transmitted which exceeds 1518 bytes (non VLAN) or 1522 bytes (VLAN) with a correct FCS value. |

### 14.5.3.6.42 Transmit Undersize Frame Counter (TUND)

Figure 14-96 describes the definition for the TUND register.

Offset eTSEC1:0x2_4728;
eTSEC3:0x2_6728

Access: Read/Write

| | 0–19 | 20–31 |
|---|---|---|
| R<br>W | — | TUND |
| Reset | All zeros | |

**Figure 14-96. Transmit Undersize Frame Counter Register Definition**

Table 14-100 describes the fields of the TUND register.

**Table 14-100. TUND Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–19 | — | Reserved |
| 20–31 | TUND | Transmit undersize frame counter. Increments for every frame less then 64 bytes, with a correct FCS value. |

### 14.5.3.6.43 Transmit Fragment Counter (TFRG)

Figure 14-97 describes the definition for the TFRG register.

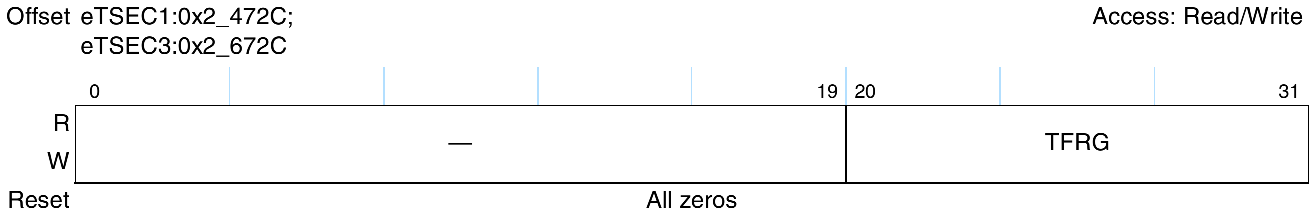


**Figure 14-97. Transmit Fragment Counter Register Definition**

Table 14-101 describes the fields of the TFRG register.

**Table 14-101. TFRG Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–19 | — | Reserved |
| 20–31 | TFRG | Transmit fragment counter. Increments for every frame less then 64 bytes, with an incorrect FCS value. |

### 14.5.3.6.44 Carry Register 1 (CAR1)

Carry register bits are cleared on carry register writes when the respective bits are set. Figure 14-98 describes the definition for the CAR1 register.

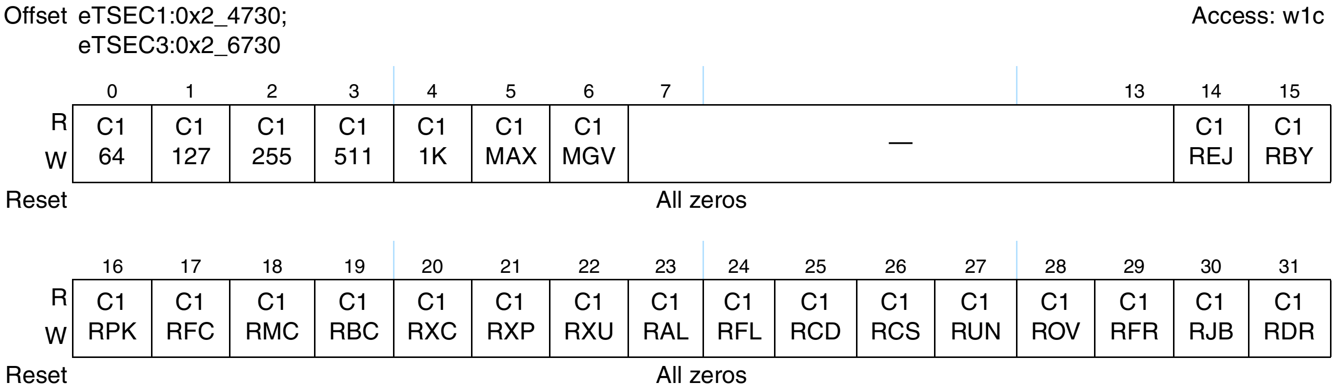


**Figure 14-98. Carry Register 1 (CAR1) Register Definition**

Table 14-102 describes the fields of the CAR1 register.

**Table 14-102. CAR1 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | C164 | Carry register 1 TR64 counter carry bit |
| 1 | C1127 | Carry register 1 TR127 counter carry bit |
| 2 | C1255 | Carry register 1 TR255 counter carry bit |
| 3 | C1511 | Carry register 1 TR511 counter carry bit |

Table 14-102. CAR1 Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 4 | C11K | Carry register 1 TR1K counter carry bit |
| 5 | C1MAX | Carry register 1 TRMAX counter carry bit |
| 6 | C1MGV | Carry register 1 TRMGV counter carry bit |
| 7–13 | — | Reserved |
| 14 | C1REJ | Carry register 1 RREJ counter carry bit |
| 15 | C1RBY | Carry register 1 RBYT counter carry bit |
| 16 | C1RPK | Carry register 1 RPKT counter carry bit |
| 17 | C1RFC | Carry register 1 RFCS counter carry bit |
| 18 | C1RMC | Carry register 1 RMCA counter carry bit |
| 19 | C1RBC | Carry register 1 RBCA counter carry bit |
| 20 | C1RXC | Carry register 1 RXCF counter carry bit |
| 21 | C1RXP | Carry register 1 RXPF counter carry bit |
| 22 | C1RXU | Carry register 1 RXUO counter carry bit |
| 23 | C1RAL | Carry register 1 RALN counter carry bit |
| 24 | C1RFL | Carry register 1 RFLR counter carry bit |
| 25 | C1RCD | Carry register 1 RCDE counter carry bit |
| 26 | C1RCS | Carry register 1 RCSE counter carry bit |
| 27 | C1RUN | Carry register 1 RUND counter carry bit |
| 28 | C1ROV | Carry register 1 ROVR counter carry bit |
| 29 | C1RFR | Carry register 1 RFRG counter carry bit |
| 30 | C1RJB | Carry register 1 RJBR counter carry bit |
| 31 | C1RDR | Carry register 1 RDRP counter carry bit |

#### 14.5.3.6.45 Carry Register 2 (CAR2)

Figure 14-99 describes the definition for the CAR2 register.

Offset eTSEC1:0x2_4734;
eTSEC3:0x2_6734

Access: w1c

| | 0 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|
| R<br>W | — | C2<br>TJB | C2<br>TFC | C2<br>TCF | C2<br>TOV |
| Reset | All zeros | | | | |

| | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | C2<br>TUN | C2<br>TFG | C2<br>TBY | C2<br>TPK | C2<br>TMC | C2<br>TBC | C2<br>TPF | C2<br>TDF | C2<br>TED | C2<br>TSC | C2<br>TMA | C2<br>TLC | C2<br>TXC | C2<br>TNC | — | C2<br>TDP |
| Reset | All zeros | | | | | | | | | | | | | | | |

**Figure 14-99. Carry Register 2 (CAR2) Register Definition**

Carry register bits are cleared on carry register write when the respective bits are set. Table 14-103 describes the fields of the CAR2 register.

**Table 14-103. CAR2 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–11 | — | Reserved |
| 12 | C2TJB | Carry register 2 TJBR counter carry bit |
| 13 | C2TFC | Carry register 2 TFCS counter carry bit |
| 14 | C2TCF | Carry register 2 TXCF counter carry bit |
| 15 | C2TOV | Carry register 2 TOVR counter carry bit |
| 16 | C2TUN | Carry register 2 TUND counter carry bit |
| 17 | C2TFG | Carry register 2 TFRG counter carry bit |
| 18 | C2TBY | Carry register 2 TBYT counter carry bit |
| 19 | C2TPK | Carry register 2 TPKT counter carry bit |
| 20 | C2TMC | Carry register 2 TMCA counter carry bit |
| 21 | C2TBC | Carry register 2 TBCA counter carry bit |
| 22 | C2TPF | Carry register 2 TXPF counter carry bit |
| 23 | C2TDF | Carry register 2 TDFR counter carry bit |
| 24 | C2TED | Carry register 2 TEDF counter carry bit |
| 25 | C2TSC | Carry register 2 TSCL counter carry bit |
| 26 | C2TMA | Carry register 2 TMCL counter carry bit |
| 27 | C2TLC | Carry register 2 TLCL counter carry bit |
| 28 | C2TXC | Carry register 2 TXCL counter carry bit |
| 29 | C2TNC | Carry register 2 TNCL counter carry bit |

**Table 14-103. CAR2 Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 30 | — | Reserved, should be cleared |
| 31 | C2TDP | Carry register 2 TDRP counter carry bit |

### 14.5.3.6.46 Carry Mask Register 1 (CAM1)

While one of the below mask bits are cleared, the corresponding carry bit in CAR1 is allowed to cause interrupt indications in register IEVENT[MSR0]. These bits all default to a set state. Figure 14-100 describes the definition for the CAM1 register.

Offset eTSEC1:0x2_4738;
eTSEC3:0x2_6738

Access: Read/Write

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | | | | | | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | M1<br>64 | M1<br>127 | M1<br>255 | M1<br>511 | M1<br>1K | M1<br>MAX | M1<br>MGV | — | | | | | | | M1<br>REJ | M1<br>RBY |
| Reset | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

| | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | M1<br>RPK | M1<br>RFC | M1<br>RMC | M1<br>RBC | M1<br>RXC | M1<br>RXP | M1<br>RXU | M1<br>RAL | M1<br>RFL | M1<br>RCD | M1<br>RCS | M1<br>RUN | M1<br>ROV | M1<br>RFR | M1<br>RJB | M1<br>RDR |
| Reset | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

**Figure 14-100. Carry Mask Register 1 (CAM1) Register Definition**

Table 14-104 describes the fields of the CAM1 register.

**Table 14-104. CAM1 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | M164 | Mask register 1 TR64 counter carry bit mask |
| 1 | M1127 | Mask register 1 TR127 counter carry bit mask |
| 2 | M1255 | Mask register 1 TR255 counter carry bit mask |
| 3 | M1511 | Mask register 1 TR511 counter carry bit mask |
| 4 | M11k | Mask register 1 TR1K counter carry bit mask |
| 5 | M1MAX | Mask register 1 TRMAX counter carry bit mask |
| 6 | M1MGV | Mask register 1 TRMGV counter carry bit mask |
| 7–13 | — | Reserved |
| 14 | M1REJ | Mask register 1 RREJ counter carry bit mask |
| 15 | M1RBY | Mask register 1 RBYT counter carry bit mask |
| 16 | M1RPK | Mask register 1 RPKT counter carry bit mask |
| 17 | M1RFC | Mask register 1 RFCS counter carry bit mask |
| 18 | M1RMC | Mask register 1 RMCA counter carry bit mask |

**Table 14-104. CAM1 Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 19 | M1RBC | Mask register 1 RBCA counter carry bit mask |
| 20 | M1RXC | Mask register 1 RXCF counter carry bit mask |
| 21 | M1RXP | Mask register 1 RXPF counter carry bit mask |
| 22 | M1RXU | Mask register 1 RXUO counter carry bit mask |
| 23 | M1RAL | Mask register 1 RALN counter carry bit mask |
| 24 | M1RFL | Mask register 1 RFLR counter carry bit mask |
| 25 | M1RCD | Mask register 1 RCDE counter carry bit mask |
| 26 | M1RCS | Mask register 1 RCSE counter carry bit mask |
| 27 | M1RUN | Mask register 1 RUND counter carry bit mask |
| 28 | M1ROV | Mask register 1 ROVR counter carry bit mask |
| 29 | M1RFR | Mask register 1 RFRG counter carry bit mask |
| 30 | M1RJB | Mask register 1 RJBR counter carry bit mask |
| 31 | M1RDR | Mask register 1 RDRP counter carry bit mask |

### 14.5.3.6.47 Carry Mask Register 2 (CAM2)

While one of the below mask bits are cleared, the corresponding carry bit in CAR2 is allowed to cause interrupt indications in register IEVENT[MSR0]. These bits default to a set state. Figure 14-101 describes the definition for the CAM2 register.

Offset eTSEC1:0x2_473C;
eTSEC3:0x2_673C

Access: Read/Write

| | 0 | | | | | | | | | | | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | | | | | | | | | | | | M2<br>TJB | M2<br>TFC | M2<br>TCF | M2<br>TOV |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

| | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | M2<br>TUN | M2<br>TFG | M2<br>TBY | M2<br>TPK | M2<br>TMC | M2<br>TBC | M2<br>TPF | M2<br>TDF | M2<br>TED | M2<br>TSC | M2<br>TMA | M2<br>TLC | M2<br>TXC | M2<br>TNC | — | M2<br>TDP |
| Reset | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |

**Figure 14-101. Carry Mask Register 2 (CAM2) Register Definition**

Table 14-105 describes the fields of the CAM2 register.

**Table 14-105. CAM2 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–11 | — | Reserved |
| 12 | M2TJB | Mask register 2 TJBR counter carry bit mask |

**Table 14-105. CAM2 Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 13 | M2TFC | Mask register 2 TFCS counter carry bit mask |
| 14 | M2TCF | Mask register 2 TXCF counter carry bit mask |
| 15 | M2TOV | Mask register 2 TOVR counter carry bit mask |
| 16 | M2TUN | Mask register 2 TUND counter carry bit mask |
| 17 | M2TFG | Mask register 2 TFRG counter carry bit mask |
| 18 | M2TBY | Mask register 2 TBYT counter carry bit mask |
| 19 | M2TPK | Mask register 2 TPKT counter carry bit mask |
| 20 | M2TMC | Mask register 2 TMCA counter carry bit mask |
| 21 | M2TBC | Mask register 2 TBCA counter carry bit mask |
| 22 | M2TPF | Mask register 2 TXPF counter carry bit mask |
| 23 | M2TDF | Mask register 2 TDFR counter carry bit mask |
| 24 | M2TED | Mask register 2 TEDF counter carry bit mask |
| 25 | M2TSC | Mask register 2 TSCL counter carry bit mask |
| 26 | M2TMA | Mask register 2 TMCL counter carry bit mask |
| 27 | M2TLC | Mask register 2 TLCL counter carry bit mask |
| 28 | M2TXC | Mask register 2 TXCL counter carry bit mask |
| 29 | M2TNC | Mask register 2 TNCL counter carry bit mask |
| 30 | — | Reserved |
| 31 | M2TDP | Mask register 2 TDRP counter carry bit mask |

### 14.5.3.6.48 Receive Filer Rejected Packet Counter (RREJ)

Figure 14-102 describes the definition for the RREJ register.

Offset eTSEC1:0x2_4740;
eTSEC3:0x2_6740

Access: Read/Write

| | 0 … 9 | 10 … 31 |
|---|---|---|
| R<br>W | — | RREJ |
| Reset | All zeros | |

**Figure 14-102. Receive Filer Rejected Packet Counter Register Definition**

Table 14-70 describes the fields of the RREJ register.

Table 14-106. RREJ Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–9 | — | Reserved |
| 10–31 | RREJ | Receive filer rejected packet counter. Increments for each frame with valid CRC received, but rejected by the receive queue filer—either due to a matching rule that asserted the REJ flag or due to filing to a RxBD ring that was not enabled (see IEVENT[FIQ] error). |

### 14.5.3.7 Hash Function Registers

This section provides detailed descriptions of the registers used for hash functions. All of the registers are 32 bits wide. The DA field of every received frame is processed through a 32-bit CRC generator (CRC-32 polynomial), and the 8 or 9 most significant bits of the CRC are mapped to a hash table entry. The user can enable a hash entry by setting its bit. A hash entry usually represents a set of addresses. A hash table hit occurs if the DA CRC result points to an enabled hash entry. Software may need to further filter the address in order to eliminate false-positive hits in the hash table.

If RCTRL[GHTX] = 0, the 8 most significant bits of the CRC are used as the hash table index. In this case, registers IGADDR0–IGADDR7 comprise a 256-entry hash table exclusively for individual (unicast) address matching, while registers GADDR0–GADDR7 comprise a 256-entry hash table for group (multicast) address matching. If RCTRL[GHTX] = 1, the group hash table is extended to all 512 entries, and the 9 most significant bits of the CRC are used as the hash table index. In this case, registers IGADDR0–IGADDR7 hold hash table entries 0–255 for group addresses, while registers GADDR0–GADDR7 hold entries 256–511 of the extended group hash table.

See Section 14.6.3.7.2, “Hash Table Algorithm,” for more information on the hash algorithm.

#### 14.5.3.7.1 Individual/Group Address Registers 0–7 (IGADDR*n*)

The IGADDR*n* registers are written by the user. Together these registers represent, depending on RCTRL[GHTX], either the 256 entries of the individual address hash table, or the first 256 entries of the extended group address hash table used in the address recognition process. The user can enable a hash entry by setting the appropriate bit. A hash table hit occurs if the DA CRC-32 result points to an enabled hash entry.

Figure 14-103 describes the definition for the IGADDR*n* register.

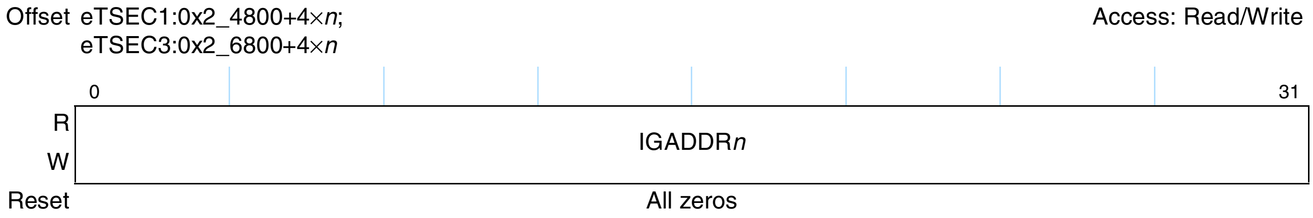


Figure 14-103. IGADDR*n* Register Definition

Table 14-108 describes the fields of the IGADDR*n* register.

**Table 14-107. IGADDR*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–31 | IGADDR*n* | Represents the 32-bit value associated with the corresponding register. When RCTRL[GHTX] = 0, IGADDR0 contains entries 0–31 of the 256-entry individual hash table and IGADDR7 represents entries 224–255. When RCTRL[GHTX] = 1, IGADDR0 contains entries 0–31 of the 512-entry extended group hash table and IGADDR7 represents entries 224–255. |

### 14.5.3.7.2 Group Address Registers 0–7 (GADDR*n*)

The GADDR*n* registers are written by the user. Together these registers represent, depending on RCTRL[GHTX], either the 256 entries of the group address hash table, or the last 256 entries of the extended group address hash table used in the address recognition process. The user can enable a hash entry by setting the appropriate bit. A hash table hit occurs if the DA CRC result points to an enabled hash entry. Figure 14-104 describes the definition for the GADDR*n* register.

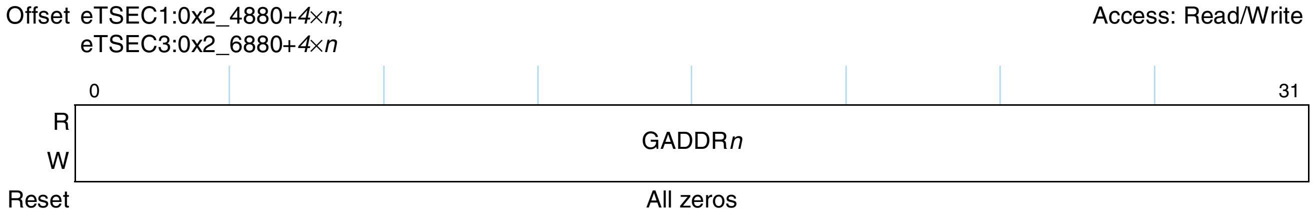


**Figure 14-104. GADDR*n* Register Definition**

Table 14-108 describes the fields of the GADDR*n* register.

**Table 14-108. GADDR*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–31 | GADDR*n* | Represents the 32-bit value associated with the corresponding register. When RCTRL[GHTX] = 0, GADDR0 contains entries 0–31 of the 256-entry group hash table and GADDR7 represents entries 224–255. When RCTRL[GHTX] = 1, GADDR0 contains entries 256–287 of the 512-entry extended group hash table and GADDR7 represents entries 480–511. |

## 14.5.3.8 FIFO Registers

This section provides detailed descriptions of the registers used to configure the FIFO interface. All of the registers are 32 bits wide. The ECNTRL[FIFM] bit is set to indicate that data transfers take place over this interface. Please refer to Section 14.6.2, “Connecting to FIFO Interfaces,” for details of the signaling protocols available.

### 14.5.3.8.1 FIFO Configuration Register (FIFOCFG)

The FIFO Configuration Register configures and enables the 8-bit FIFO interface.

Figure 14-105 describes the definition for the FIFOCFG register.

Offse eTSEC1:0x2_4A00;
t eTSEC3:0x2_6A00

Access: Read/Write

| | 0 | | | | | | | 7 | 8 | | | | | | | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | | | | | | | | IPG | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 16 | 17 | 18 | 19 | 20 | | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | RRX | RTX | RXE | TXE | — | | | LPB | RFC | TFC | FFC | CRCAPP | CRCCHK | — | SIGM | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

**Figure 14-105. FIFOCFG Register Definition**

Table 14-109 describes the fields of the FIFOCFG register.

**Table 14-109. FIFOCFG Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–7 | — | Reserved |
| 8–15 | IPG | Minimum inter packet gap. This sets the minimum number of cycles inserted between back-to-back frames transmitted over the FIFO interface. The minimum required is 3 cycles if CRCAPP=0 and 7 cycles for 8-bit interfaces if CRCAPP=1. |
| 16 | RRX | Enable reset of FIFO receive function.<br>0 Do not reset the FIFO receiver.<br>1 Reset the FIFO receiver for as long as this bit is set. |
| 17 | RTX | Enable reset of FIFO transmit function.<br>0 Do not reset the FIFO transmitter.<br>1 Reset the FIFO transmitter for as long as this bit is set. |
| 18 | RXE | Enable FIFO receive function.<br>0 Disable reception over the FIFO interface, ignoring data presented to the signals.<br>1 Enable normal reception over the FIFO interface. |
| 19 | TXE | Enable FIFO transmit function.<br>0 Disable transmission over the FIFO interface.<br>1 Enable normal transmission over the FIFO interface. |
| 20–22 | — | Reserved. |
| 23 | LPB | Loopback enable.<br>0 Do not loopback data in the FIFO interface.<br>1 Loopback transmitted data to the FIFO receiver rather than outputting transmitted data to signals. |
| 24 | RFC | Enable receive flow control. Setting FFC overrides this bit.<br>0 Do not allow the FIFO receiver to assert link-level flow control if eTSEC requires it.<br>1 Allow the FIFO receiver to assert link-level flow control if eTSEC requires it. This is the default setting. |
| 25 | TFC | Enable transmit flow control.<br>0 Do not allow the FIFO transmitter to assert link-level flow control if transmit data is unavailable, resulting in underruns.<br>1 Allow the FIFO transmitter to assert link-level flow control if transmit data is unavailable and SIGM = 01. This is the default setting. |

**Table 14-109. FIFOCFG Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 26 | FFC | Force flow control. This can be used by software to stop reception on the FIFO interface.<br>0 Do not assert link-level flow control through the RXFC signal unless eTSEC requires flow control.<br>1 Force flow control on the RXFC signal in encoded FIFO packet mode regardless of eTSEC pause requirements. |
| 27 | CRCAPP | Append a CRC (CRC-32 algorithm, as per IEEE Std. 802.3) to the end of every transmitted frame.<br>0 Do not automatically append a CRC to transmitted frames. Allow TxBD[TC], if set, to control when a CRC is appended.<br>1 Automatically append a CRC to transmitted frames. Ignore TxBD[TC]. |
| 28 | CRCCHK | Check the CRC (CRC-32 algorithm, as per IEEE Std. 802.3) at the end of every frame.<br>0 Do not automatically check the last 4 bytes of received frames for a valid CRC.<br>1 Automatically check the last 4 bytes of received frames for a valid CRC. If a CRC error is detected, or insufficient data is received to recover the CRC, the RxBD[CR] bit is set. |
| 29 | — | Reserved |
| 30–31 | SIGM | FIFO signaling mode. Determines how the GMII signals are interpreted as framing signals.<br>00 GMII style mode.<br>01 Encoded packet mode.<br>10 Reserved<br>11 Reserved |

### 14.5.3.9 DMA Attribute Registers

This section describes the two eTSEC DMA attribute registers.

#### 14.5.3.9.1 Attribute Register (ATTR)

The attribute register defines memory access attributes and transaction types used to access buffer descriptors, to write receive data, and to read transmit data. Snoop enable attributes may be set for reading buffer descriptors and for reading transmit data. Buffer descriptors may be written with attributes that cause allocation into the L2 cache. Similarly, broad sections of a receive frame header may have attributes attached that cause allocation in the L2 cache. This process of specifying a region of each frame to stash into the L2 cache is referred to as *extraction*, which is specified in conjunction with register ATTRELI. ATTR[ELCWT] only has meaning if ATTRELI[EL] is non-zero. It is important to note that even though portions of received frames may be stashed to L2 cache, this is only a performance optimization as entire frames are still written to off-chip memory regardless of settings in ATTR.

Figure 14-106 describes the definition for the ATTR register.

Offset eTSEC1:0x2_4BF8;
eTSEC3:0x2_6BF8

Access: Read/Write

| | 0 … 16 | 17 18 | 19 | 20 21 | 22 23 | 24 | 25 | 26 … 31 |
|---|---|---|---|---|---|---|---|---|
| R<br>W | — | ELCWT | — | BDLWT | — | RDSEN | RBDSEN | — |
| Reset | All zeros | | | | | | | |

**Figure 14-106. ATTR Register Definition**

Table 14-110 describes the fields of the ATTR register.

Table 14-110. ATTR Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–16 | — | Reserved |
| 17–18 | ELCWT | Extracted L2 cache write type. Specifies the write transaction type to perform for the extracted data. For maximum performance, it is recommended that if ELCWT is set to allocate, BDLWT should also be set to allocate.Writes to cache are always performed with snoop.<br>00 No allocation performed.<br>01 Reserved<br>10 Allocate L2 cache line.<br>11 Reserved. |
| 19 | — | Reserved |
| 20–21 | BDLWT | Buffer descriptor L2 cache write type. specifies the write transaction type to perform for the bufferdescriptor for a receive frame. Writes to cache are always performed with snoop.<br>00 No allocation performed.<br>01 Reserved<br>10 Allocate L2 cache line.<br>11 Reserved. |
| 22–23 | — | Reserved |
| 24 | RDSEN | Rx data snoop enable.<br>0 Disables snooping of all receive frames data to memory unless ELCWT specifies L2allocation.<br>1 Enables snooping of all receive frames data to memory. |
| 25 | RBDSEN | RxBD snoop enable.<br>0 Disables snooping of all receive BD memory accesses unless BDLWT specifies L2 allocation.<br>1 Enables snooping of all receive BD memory accesses. |
| 26–31 | — | Reserved |

### 14.5.3.9.2 Attribute Extract Length and Extract Index Register (ATTRELI)

The ATTRELI registers are written by the user to specify the extract index and extract length for extracting received frames. The extract length is typically set to the expected length of extracted packet headers. Figure 14-107 describes the definition for the ATTRELI register.

Offset eTSEC1:0x2_4BFC;
eTSEC3:0x2_6BFC

Access: Read/Write

| | 0 1 | 2 12 | 13 17 | 18 25 | 26 31 |
|---|---|---|---|---|---|
| R<br>W | — | EL | — | EI | — |
| Reset | All zeros | | | | |

Figure 14-107. ATTRELI Register Definition

Table 14-111 describes the fields of the ATTRELI register.

**Table 14-111. ATTRELI Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–1 | — | Reserved |
| 2–12 | EL | Extracted length. Specifies the number of bytes, as a multiple of 8 bytes, to extract from the receive frame. The DMA controller uses this field to perform extraction. If cleared, no extraction is performed. |
| 13–15 | — | To ensure that EL is a multiple of 8 bytes, these bits should be written with zero. |
| 16–17 | — | Reserved |
| 18–25 | EI | Extracted index. Points to the first byte, as a multiple of 64 bytes, within the receive frame as sent to memory from which to begin extracting data. |
| 26–31 | — | To ensure that EI is a multiple of 8 bytes, these bits should be written with zero. |

## 14.5.3.10 Lossless Flow Control Configuration Registers

When enabled through RCTRL[LFC], the eTSEC tracks location of the last free BD in each Rx BD ring through the value of RFBPTR*n*. Using this pointer and the ring length stored in RQPRM*n*[LEN], the eTSEC continuously calculates the number of free BDs in the ring. Whenever the calculated number of free BDs in the ring drops below the pause threshold specified in RQPRM*n*[FBTHR], the eTSEC issues link layer flow control. It continues to assert flow control until the free BD count for each active ring reaches or exceeds RQPRM*n*[FBTHR]. See section 14.6.6.1/14-191 for the theory of operation of these registers.

### 14.5.3.10.1 Receive Queue Parameters 0–7 (RQPRM0–PQPRM7)

The RQPRM*n* registers specify the minimum number of BDs required to prevent flow control being asserted and the total number of Rx BDs in their respective ring. Whenever the free BD count calculated by the eTSEC for any active ring drops below the value of RQPRM*n*[FBTHR] for that ring, link level flow control is asserted. Software must not write to RQPRM*n* while LFC is enabled and the eTSEC is actively receiving frames. However, software may modify these registers after disabling LFC by clearing RCTRL[LFC]. Note that packets may be lost due to lack of RxBDs while RCTRL[LFC] is clear. Software can prevent packet loss by manually generating pause frames (through TCTRL[TFC_PAUSE]) to cover the time when RCTRL[LFC] is clear. Figure 14-108 describes the definition for the RQPRM*n* register.

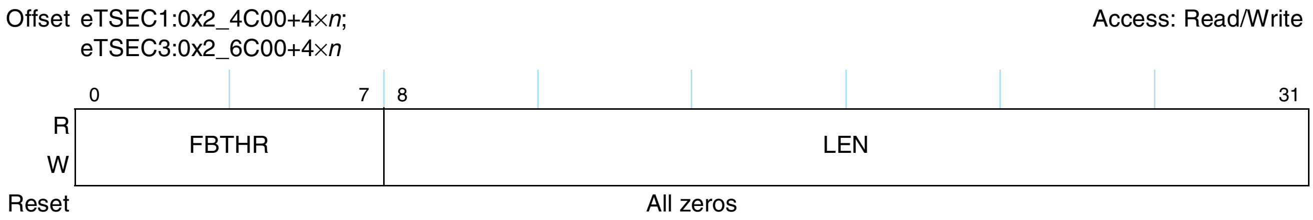


**Figure 14-108. RQPRM Register Definition**

Table 14-112 describes the fields of the RQPRM register.

**Table 14-112. RQPRM Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–7 | FBTHR | Free BD threshold. Minimum number of BDs required for normal operation. If the eTSEC calculated number of free BDs drops below this threshold, link layer flow control is asserted. |
| 8–31 | LEN | Ring length. Total number of Rx BDs in this ring. |

### 14.5.3.10.2 Receive Free Buffer Descriptor Pointer Registers 0–7 (RFBPTR0–RFBPTR7)

The RFBPTR*n* registers specify the location of the last free buffer descriptor in their respective ring. These registers live in the same 32b address space – and must share the same 4 most significant bits – as RBPTR*n*. That is, RFBPTR*n* and its associated RBPTR*n* must remain in the same 256MB page. Like RBPTR*n*, whenever RBASE*n* is updated, RFBPTR*n* is initialized to the value of RBASE*n*. This indicates that the ring is completely empty. As buffers are freed and their respective BDs are returned (by setting the EMPTY bit) to the ring, software is expected to update this register. The eTSEC then performs modulo arithmetic involving RBASE*n*, RBPTR*n* and RFBPTR*n* to determine the number of free BDs remaining in the ring. If, at any stage, the value written to RFBPTR*n* matches that of the respective RBPTR*n* the eTSEC free BD calculation assumes that the ring is now completely empty. For more information on the recommended use of these registers, see Section 14.6.6.1, “Back Pressure Determination through Free Buffers.”
Figure 14-109 describes the definition for the RFBPTR*n* register.

Offset eTSEC1:0x2_4C44+8×*n*;
eTSEC3:0x2_6C44+8×*n*

Access: Read/Write

| | 0 … 28 | 29 … 31 |
|---|---|---|
| R<br>W | RFBPTR*n* | — |
| Reset | All zeros | |

**Figure 14-109. RFBPTR0–RFBPTR7 Register Definition**

Table 14-113 describes the fields of the RFBPTR*n* registers.

**Table 14-113. RFBPTR0–RFBPTR7 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–28 | RFBPTR | Pointer to the last free BD in RxBD Ring *n*. When RBASE*n* is updated, eTSEC initializes RFBPTR*n* to the value in the corresponding RBASE*n*.<br>Software may update this register at any time to inform the eTSEC the location of the last free BD in the ring. Note that the 3 least-significant bits of this register are read only and zero. |
| 29–31 | — | Reserved. |

### 14.5.3.11 Hardware Assist for IEEE1588 Compliant Timestamping

IEEE 1588 compliant timestamping on this device is accomplished using the per-port transmit timestamping registers within each Ethernet controller memory space (See Section 14.5.3.2.12, “Transmit Time Stamp Identification Register (TMR_TXTS1–2_ID),” and Section 14.5.3.2.13, “Transmit Time Stamp Register (TMR_TXTS1–2_H/L).”) in conjunction with the following common registers, which are located within the memory space for eTSEC1. Because the common 1588 timestamping registers exist within the eTSEC1 memory space, the eTSEC1 controller must remain enabled in order to use 1588 timestamping for any Ethernet port.

#### 14.5.3.11.1 Timer Control Register (TMR_CTRL)

This register is used to reset, configure, and initialize the eTSEC precision timer clock. The control of all timer function is performed via programming eTSEC1.The register in eTSEC1 is shared for all eTSECs. Figure 14-7 describes the definition for the TMR_CTRL register.

Register fields not described below are reserved.

Offset eTSEC1:0x2_4E00 Access: Mixed

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | | | | | | | | | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | ALM1P | ALM2P | — | FS | PP1L | PP2L | TCLK_PERIOD | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

| | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | RTPE | FRD | — | — | ESFDP | ESFDE | ETEP2 | ETEP1 | COPH | CIPH | TMSR | — | BYP | TE | CKSEL | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

**Figure 14-110. TMR_CTRL Register Definition**

Table 14-114 describes the fields of the TMR_CTRL register. Register fields not described below are reserved.

**Table 14-114. TMR_CTRL Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | ALM1P | Alarm1 output polarity<br>0 active high output<br>1 active low output |
| 1 | ALM2P | Alarm2 output polarity<br>0 active high output<br>1 active low output |
| 3 | FS | FIPER start indication<br>0 Fiper is enabled through timer enable<br>1 Fiper is enabled through timer enable and alarm indication. |
| 4 | PP1L | Fiper1 pulse loopback mode enabled.<br>0 Trigger1 input is based upon normal external trigger input.<br>1 Fiper1 pulse is looped back into Trigger1 input. |

**Table 14-114. TMR_CTRL Register Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 5 | PP2L | Fiper2 pulse loopback mode enabled.<br>0 Trigger2 input is based upon normal external trigger input.<br>1 Fiper2 pulse is looped back into Trigger2 input. |
| 6–15 | TCLK_ PERIOD | 1588 timer reference clock period. The timer clock counter will increment by TCLK_PERIOD every time the accumulator register overflows. This clock period must be larger than the clock period of the timer reference clock. For applications where user does not want the clock period to be added, they can program this field to 1 to count the clock ticks. This field defaulted to 1 to count overflow ticks.<br>For nanosecond granularity on 1588 timer counter rate, the TCLK_PERIOD should be calculated using the following equation:<br>TCLK_PERIOD = $10^9$/Nominal_Frequency |
| 16 | RTPE | Record Tx Time-Stamp to PAL Enable.<br>When set, and FCB[PTP] is set, the 8-byte time-stamp for the packet is written to the PAL located in external memory location at an offset of 16 bytes from the start of the Data Buffer Pointer of the first TxBD. For guidelines on using the RTPE bit, refer to Section 14.6.7.5, “Time-Stamp Insertion on Transmit Packets.” |
| 17 | FRD | FIPER Realignment Disable<br>0 Fiper Realignment is enabled.<br>1 Fiper Realignment is disabled. |
| 18–19 | — | Reserved |
| 20 | ESFDP | External Tx/Rx SFD Polarity.<br>0 Time stamp on rising edge of external SFD indication.<br>1 Time stamp on falling edge of external SFD indication. |
| 21 | ESFDE | External Tx/Rx SFD Enable.<br>0 Time stamp PTP TX frame based on MAC’s SFD indication.<br>1 Time stamp PTP TX frame based on external SFD indication from PHY. |
| 22 | ETEP2 | External trigger 2 edge polarity<br>0 Time stamp on the rising edge of the external trigger<br>1 Time stamp on the falling edge of the external trigger |
| 23 | ETEP1 | External trigger 1 edge polarity<br>0 time stamp on the rising edge of the external trigger<br>1 time stamp on the falling edge of the external trigger |
| 24 | COPH | Generated clock (TSEC_1588_CLK_OUT) output phase.<br>0 non-inverted divided clock is output<br>1 inverted divided clock is output |
| 25 | CIPH | Oscillator input clock phase.<br>0 non-inverted timer input clock<br>1 inverted timer input clock (NOTE: this setting is reserved if CKSEL=01.) |
| 26 | TMSR | Timer soft reset. When enabled, it resets all the timer registers and state machines.<br>0 normal operation<br>1 place entire timer in reset except control and config registers<br>NOTE: Prior to initiating timer reset (setting TMSR), must gracefully stop receiver (See MACCFG1[RX_EN] description).<br>User programmable registers are not reset by the soft reset e.g. TMR_CTRL, TMR_TEMASK, TMR_PEMASK, TMR_ADD, TMR_PRSC, TMROFF_H/L, TMR_ALARMn, and TMR_FIPERn. |
| 27 | — | Reserved |

**Table 14-114. TMR_CTRL Register Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 28 | BYP | Bypass drift compensated clock<br>0 64-bit clock counter is incremented on the accumulator overflow<br>1 64-bit clock counter is directly driven from the external oscillator ignoring accumulator overflow |
| 29 | TE | 1588 timer enable. If not enabled, all the timer registers and state machines are disabled.<br>0 timer not enabled<br>1 timer enabled and resume normal operation |
| 30–31 | CKSEL | 1588 Timer reference clock source select.<br>00 External high precision timer reference clock (TSEC_1588_CLK_IN)<br>01 eTSEC system clock<br>10 eTSEC1 transmit clock<br>11 RTC clock input Note that the 1588 reference clock must be no slower than 1/7 the Rx_clk frequency.<br>The default clock select is eTSEC system clock, which is always active when eTSEC is enabled. The user must ensure the corresponding clock source is active before changing the 1588 refclk selection to external reference, RTC, or TX clock. Selecting an inactive 1588 reference clock may cause boundedly undefined behavior in the ethernet controller and on accesses to the 1588 registers. |

### 14.5.3.11.2 Timer Event Register (TMR_TEVENT)

The eTSEC precision timer implementation can generate additional interrupts that are independent of the frame based events that controlled via IEVENT. The timer interrupts are not affected by any interrupt coalescing that may be specified in TXIC/RXIC. Software may poll this register at any time to check for pending interrupts. If an event occurs and its corresponding enable bit is set in the event mask register (TEMASK), the event also causes a hardware interrupt at the PIC. A bit in the timer event register is cleared by writing a 1 to that bit position. Figure 14-4 describes the definition for the TMR_TEVENT register.

Offset eTSEC1:0x2_4E04 Access: w1c

| | 0 5 | 6 | 7 | 8 13 | 14 | 15 |
|---|---|---|---|---|---|---|
| R<br>W | — | ETS2 | ETS1 | — | ALM2 | ALM1 |
| Reset | All zeros | | | | | |

| | 16 23 | 24 | 25 | 26 | 27 31 |
|---|---|---|---|---|---|
| R<br>W | — | PP1 | PP2 | PP3 | — |
| Reset | All zeros | | | | |

**Figure 14-111. TMR_TEVENT Register Definition**

Table 14-115 describes the fields of the TMR_TEVENT register fields for the timer.

Table 14-115. TMR_TEVENT Register Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–6 | — | Reserved |
| 6 | ETS2 | External trigger 2 timestamp sampled<br>0 external trigger timestamp not sampled<br>1 external trigger timestamp sampled |
| 7 | ETS1 | External trigger 1 timestamp sampled<br>0 external trigger timestamp not sampled<br>1 external trigger timestamp sampled |
| 8–13 | — | Reserved |
| 14 | ALM2 | Current time equaled alarm time register 2<br>0 alarm time has not be reached yet<br>1 alarm time has been reached |
| 15 | ALM1 | Current time equaled alarm time register 1<br>0 alarm time has not be reached yet<br>1 alarm time has been reached |
| 16–23 | — | Reserved |
| 24 | PP1 | Indicates that a periodic pulse has been generated based on FIPER1 register.<br>0 periodic pulse not generated<br>1 periodic pulse generated |
| 25 | PP2 | Indicates that a periodic pulse has been generated based on FIPER2 register.<br>0 periodic pulse not generated<br>1 periodic pulse generated |
| 26 | PP3 | Indicates that a periodic pulse has been generated based on FIPER3 register.<br>0 periodic pulse not generated<br>1 periodic pulse generated |
| 27–31 | — | Reserved |

#### 14.5.3.11.3 Timer Event Mask Register (TMR_TEMASK)

Timer event mask register. The event mask register provides control over which possible interrupt events in the TMR_TEVENT register are permitted to participate in generating hardware interrupts to the PIC. All implemented bits in this register are R/W and cleared upon a hardware reset. Figure 14-116 describes the definition for the TMR_TEMASK register.

Offset eTSEC1:0x2_4E08 Access: Read/Write

| | 0–5 | 6 | 7 | 8–13 | 14 | 15 |
|---|---|---|---|---|---|---|
| R<br>W | — | ETS2EN | ETS1EN | — | ALM2EN | ALM1EN |
| Reset | All zeros | | | | | |

| | 16–23 | 24 | 25 | 26–31 |
|---|---|---|---|---|
| R<br>W | — | PP1EN | PP2EN | — |
| Reset | All zeros | | | |

**Table 14-116. TMR_TEMASK Register Definition**

Table 14-117 describes the fields of the TMR_TEMASK register fields for the timer.

**Table 14-117. TMR_TEMASK Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–5 | — | Reserved |
| 6 | ETS2EN | External trigger 2 timestamp sample event enable |
| 7 | ETS1EN | External trigger 1 timestamp sample event enable |
| 8–13 | — | Reserved |
| 14 | ALM2EN | Timer ALM1 event enable |
| 15 | ALM1EN | Timer ALM2 event enable |
| 16–23 | — | Reserved |
| 24 | PP1EN | Periodic pulse event 1 enable |
| 25 | PP2EN | Periodic pulse event 2 enable |
| 26–31 | — | Reserved |

#### 14.5.3.11.4 Timer PTP Packet Event Register (TMR_PEVENT)

The eTSEC precision timer logic can generate interrupts upon the capture of a timestamp due to either transmission or reception of a frame. If an event occurs and its corresponding enable bit is set in the event mask register (PEMASK), the event also causes a hardware interrupt at the PIC. A bit in the timer event register is cleared by writing a 1 to that bit position. Figure 14-112 describes the definition for the TMR_PEVENT register.

Offset eTSEC1:0x2_4E0C Access: Read/Write

| | 0–15 |
|---|---|
| R<br>W | — |
| Reset | All zeros |

| | 16–21 | 22 | 23 | 24–30 | 31 |
|---|---|---|---|---|---|
| R<br>W | — | TXP2 | TXP1 | — | RXP |
| Reset | All zeros | | | | |

**Figure 14-112. TMR_PEVENT Register Definition**

Table 14-118 describes the fields of the TMR_PEVENT register fields for the timer.

**Table 14-118. TMR_PEVENT Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–21 | — | Reserved |
| 22 | TXP2 | Indicates that a PTP frame has been transmitted and its timestamp is stored in TXTS2 register.<br>0 PTP packet not transmitted<br>1 PTP packet has been transmitted |
| 23 | TXP1 | Indicates that a PTP frame has been transmitted and its timestamp is stored in TXTS1 register.<br>0 PTP packet not transmitted<br>1 PTP packet has been transmitted |
| 24–30 | — | Reserved |
| 31 | RXP | Indicates that a PTP frame has been received<br>0 PTP packet not received<br>1 PTP packet has been received |

### 14.5.3.11.5 Timer Event Mask Register (TMR_PEMASK)

Timer event mask register. The event mask register provides control over which possible interrupt events in the TMR_PEVENT register are permitted to participate in generating hardware interrupts to the PIC. All implemented bits in this register are R/W and cleared upon a hardware reset. Figure 14-113 describes the definition for the TMR_PEMASK register.

Offset eTSEC1:0x2_4E10 Access: Read/Write

| | 0–15 |
|---|---|
| R<br>W | — |
| Reset | All zeros |

| | 16–21 | 22 | 23 | 24–30 | 31 |
|---|---|---|---|---|---|
| R<br>W | — | TXP2EN | TXP1EN | — | RXPEN |
| Reset | All zeros | | | | |

**Figure 14-113. TMR_PEMASK Register Definition**

Table 14-119 describes the fields of the TMR_PEMASK register fields for the timer.

Table 14-119. TMR_PEMASK Register Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–21 | — | Reserved |
| 22 | TXP2EN | Transmit PTP packet event 2 enable |
| 23 | TXP1EN | Transmit PTP packet event 1 enable |
| 24–30 | — | Reserved |
| 31 | RXPEN | Receive PTP packet event enable |

### 14.5.3.11.6 Timer Status Register (TMR_STAT)

This register requires the eTSEC filer to be enabled (via RCTRL[FILREN]). When eTSEC generates an interrupt based on the timestamp event for a received packet, the queue ID which the incoming packet will be sent to is captured in this register. This register update is synchronized with the RXF interrupt of the corresponding received packet. Writing 1 to any bit of this register clears it. Figure 14-120 describes the definition for the TMR_STAT register.

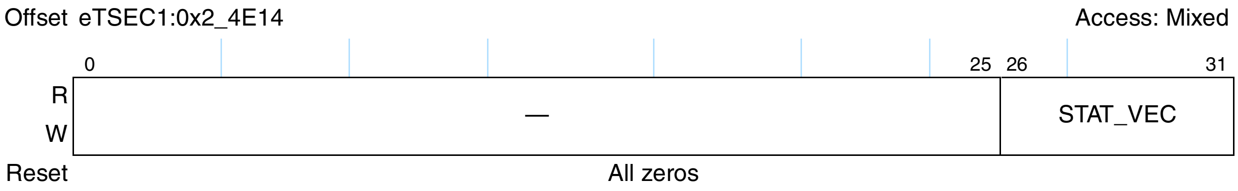


Table 14-120. TMR_STAT Register Definition

Table 14-121 describes the fields of the TMR_STAT register.

Table 14-121. TMR_STAT Register Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–25 | — | Reserved |
| 26–31 | STAT_VEC | Timer general purpose status vector. It will store the 6-bit queue number generated by the filer. User to decode this status vector. For example, user can encode received PTP packet message types (Sync, Delay_req, Follow_up, Delay_resp, Management) in the filer virtual queue field. |

### 14.5.3.11.7 Timer Counter Register (TMR_CNT_H/L)

The timer register (TMR_CNT_H/L) represents accurate time in terms clock ticks or in nano-seconds. Writes to these registers will override the previous time. The register in eTSEC1 is shared for all eTSECs. This is a read/write register. Figure 14-114 describes the definition for the TMR_CNT_H/L register.

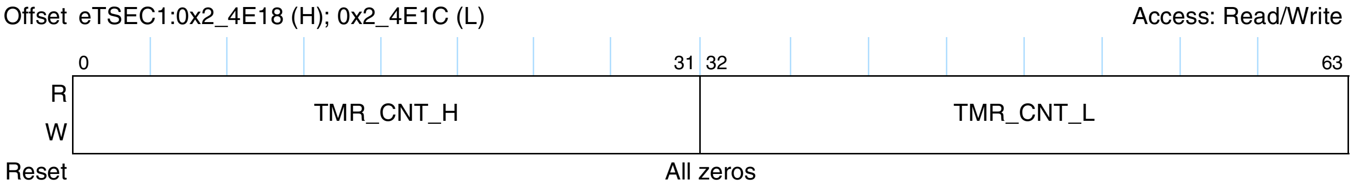


**Figure 14-114. TMR_CNT_H Register Definition**

Table 14-122 describes the fields of the TMR_CNT_H/L register.

**Table 14-122. TMR_CNT_H/L Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–63 | TMR_CNT_ H/L | Value of the current time counter. Current time is calculated by adding TMROFF_H/L with the TMR_CNT_H/L counter. This register can be written through the register writes.Writes to the TMR_CNT_L register copies the written value into the shadow TMR_CNT_L register. Writes to the TMR_CNT_H register copies the values written into the shadow TMR_CNT_H register. Contents of the shadow registers are copied into the TMR_CNT_L and TMR_CNT_H registers following a write into the TMR_CNT_H register. Writes to these registers have precedence over the timer increment. The user must write to TMR_CNT_L register first.<br>Reads from the TMR_CNT_L register copies the entire 64-bit clock time of the read enable into the TMR_CNT_H/L shadow registers. Read instruction from the TMR_CNT_H register reads the value stored in the TMR_CNT_H shadow register. The user must read the TMR_CNT_L register first to get correct 64-bit TMR_CNT_H/L counter values. |

### 14.5.3.11.8 Timer Drift Compensation Addend Register (TMR_ADD)

Timer drift compensation addend register (TMR_ADD) is used to hold timer frequency compensation value (FreqCompensationValue). The nominal frequency of the clock counter is determined by the FreqDivRatio and the clock frequency (FreqClock). This register is programmed with $2^{32}$/FreqDivRatio. Frequency division ratio (FreqDivRatio) is the ratio between the frequency of the oscillator (TimerOsc) and the desired clock frequency (NominalFreq). FreqDivRatio is a design constant chosen to be greater than 1.0001. The ADDEND value is added to the 32-bit accumulator register at every rising edge of the oscillator clock (TimerOsc). The clock counter is incremented at every carry pulse of the accumulator. Only one of this register is required for the entire group of eTSECs. Figure 14-115 describes the definition of the TMR_ADD register.

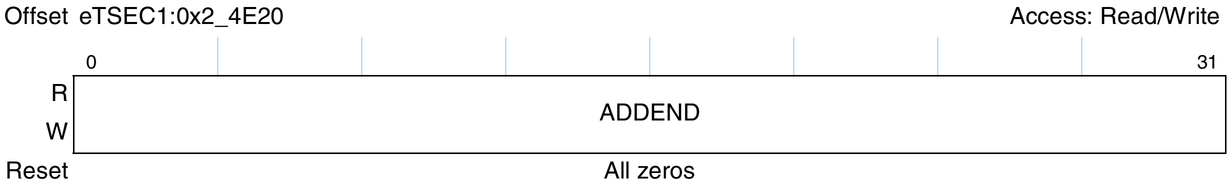


**Figure 14-115. TMR_ADD Register Definition**

Table 14-123 describes the fields of the TMR_ADD register fields for the timer.

Table 14-123. TMR_ADD Register Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–31 | ADDEND | Timer drift compensation addend register value. It is programmed with a value of 2^32/FreqDivRatio.<br>For example,<br>TimerOsc = 50 MHz<br>NominalFreq = 40 MHz<br>FreqDivRatio = 1.25<br>ADDEND = ceil(2^32/1.25) = 0xCCCC_CCCD |

#### 14.5.3.11.9 Timer Accumulator Register (TMR_ACC)

Timer accumulator register accumulates the value of the addend register into it. An overflow pulse of the accumulator is used to increment the timer clock by TMR_CTRL[TCLK_PERIOD]. This register is read only in normal operation. The register in eTSEC1 is shared for all eTSECs. Figure 14-116 describes the definition of the TMR_ACC register.

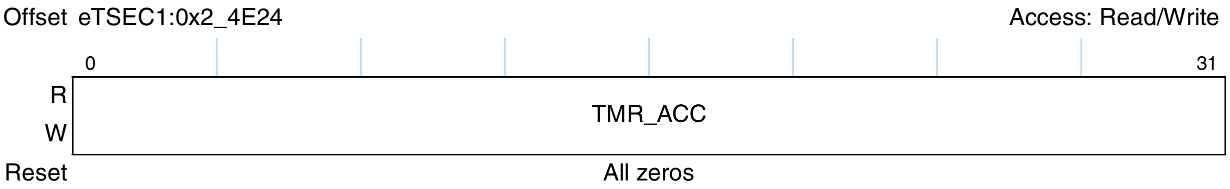


Figure 14-116. TMR_ACC Register Definition

Table 14-124 describes the fields of the TMR_ACC register.

Table 14-124. TMR_ACC Register Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–31 | TMR_ACC | 32-bit timer accumulator register |

#### 14.5.3.11.10 Timer Prescale Register (TMR_PRSC)

Timer generated output clock prescale register. It is used to adjust output clock frequency that is put onto the 1588 clock output signal. The register in eTSEC1 is shared for all eTSECs. Figure 14-117 describes the definition for the TMR_PRSC register.

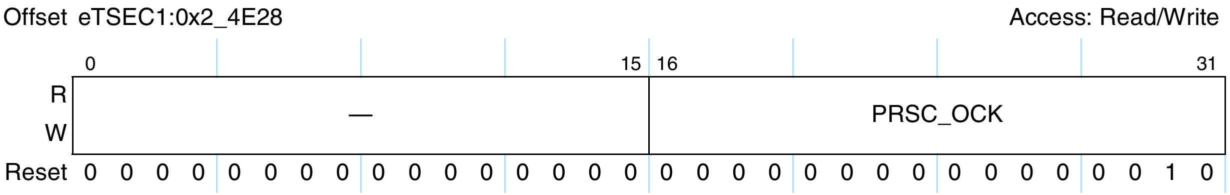

Offset eTSEC1:0x2_4E28
Access: Read/Write
0
15
16
31
R
W
—
PRSC_OCK
Reset 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0

Figure 14-117. TMR_PRSC Register Definition

Table 14-125 describes the fields of the TMR_PRSC register.

Table 14-125. TMR_PRSC Register Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–15 | — | Reserved |
| 16–31 | PRSC_OCK | Output clock division/prescale factor. Output clock is generated by dividing the timer input clock by this number. Programmed value in this field must be greater than 1. Any value less than 1 is treated as 2. |

### 14.5.3.11.11 Timer Offset Register (TMROFF_H/L)

The timer offset register is used to provide current time by adding its value to the clock counter. Figure 14-118 describes the definition of the TMROFF_H/L register.

**NOTE**

> All TMROFF_H registers in a device should be set to the same value, and all TMROFF_L registers in a device should be set to the same value. Otherwise, the precision time protocol may not work.

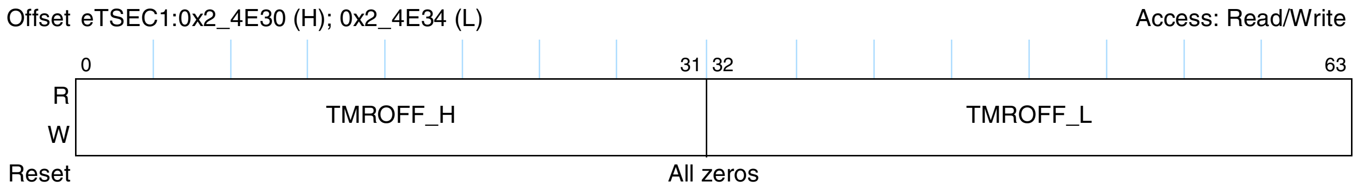


Figure 14-118. TMROFF_H/L Register Definition

Table 14-126 describes the fields of the TMROFF_H/L register.

Table 14-126. TMROFF_H/L Register Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–63 | TMROFF_H/L | Offset value of the clock counter. Current time in is calculated by adding TMROFF_H/L with the timer's counter TMR_CNT_H/L register. |

### 14.5.3.11.12 Alarm Time Comparator Register (TMR_ALARM1–2_H/L)

Alarm time comparator register (TMR_ALARM*n*_H/L). This register holds alarm time for comparison with the current time counter. There are two of these registers for eTSEC1 which are shared amongst all eTSECs. Figure 14-119 describes the definition for the TMR_ALARM*n*_H/L register.

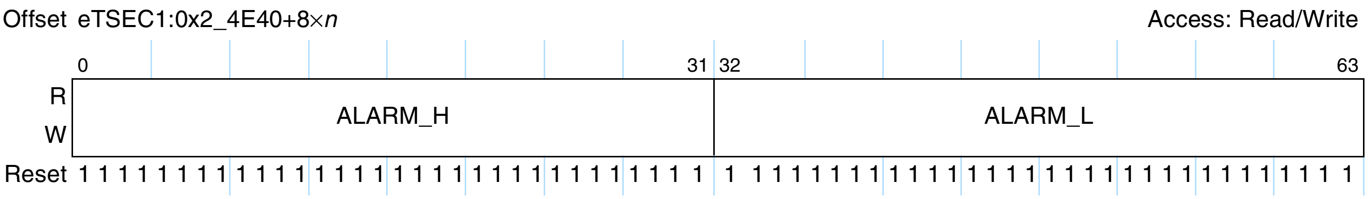


Figure 14-119. TMR_ALARM1-2_H/L Register Definition

Table 14-127 describes the fields of the TMR_ALARM*n*_H/L register.

**Table 14-127. TMR_ALARM*n*_H/L Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–63 | ALARM_H/L | Alarm time comparator register. The corresponding alarm event in TMR_TEVENT is set when the current time counter becomes equal to or greater than the alarm time compare value in TMR_ALARM*n*_L/H. Writing the TMR_ALARM*n*_L register deactivates the alarm event after it has fired. Writing the TMR_ALARM*n*_L followed by the TMR_ALARM*n*_H register rearms the alarm function with the new compare value.<br>The value programmed in this register must be an integer multiple of TMR_CTRL[TCLK_PERIOD] in order to get correct result. This register is reset to all ones to avoid false alarm after reset.<br>In FS mode the alarm trigger is used as an indication to the fiper start down counting. Only alarm 1 supports this mode. In FS mode, alarm polarity bit should be configured to 0 (rising edge). |

### 14.5.3.11.13 Timer Fixed Interval Period Register (TMR_FIPER1–3)

Timer fixed interval period pulse generator register. It is used to generate periodic pulses. This register is reset with 0xFFFF_FFFF to prevent any false pulse upon initialization. The down count register loads the value programmed in the fixed period interval (FIPER). FIPER register must be programmed before the timer is enabled. At every tick of the timer accumulator overflow, the counter decrements by the value of TMR_CTRL[TCLK_PERIOD]. It generates a pulse when the down counter value reaches zero. It reloads the down counter in the cycle following a pulse.

Should a user wish to use the TMR_FIPER1 register to generate a 1 PPS event, the following setup should be used:

- Program TMR_FIPER1 to a value that will generate a pulse every second,
- Program TMR_ALARM1 to the correct time for the first PPS event
- Enable the timer

The eTSEC will then wait for TMR_ALARM1 to expire before enabling the count down of TMR_FIPER1. The end result will be that TMR_FIPER1 will pulse every second after the original timer ALARM1 expired.

Note:

In the case where the PPS signals are required to be phased aligned to the prescale output clock, the alarm value should be configured to **1 clock period less** than the wanted value.

In order to keep tracking the prescale output clock, each time before enabling the FIPER, the user must reset the FIPER by writing a new value to the register. The ratio between the prescale register value and the FIPER value should be devisable by the clk period.

FIPER_VALUE = (prescale_value × tclk_per × N) – tclk_per

For example:

prescale = 9

clock period = 10

The FIPER can get the following values: 80, 170, 260 .......

The three registers in eTSEC1 are shared for all eTSECs. Figure 14-120 describes the definition for the TMR_FIPER register.

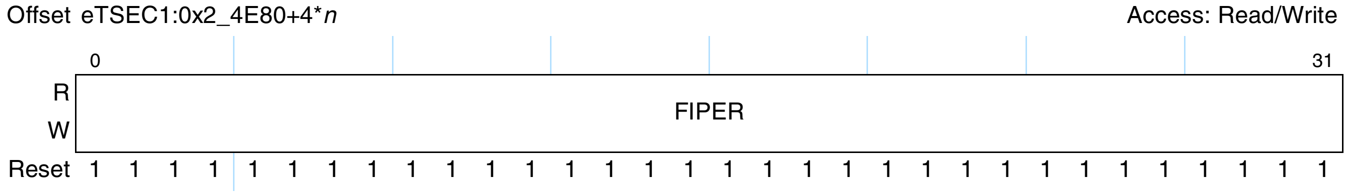

Offset eTSEC1:0x2_4E80+4*n
Access: Read/Write
0
31
R
W
FIPER
Reset 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

**Figure 14-120. TMR_FIPER*n* Register Definition**

Table 14-128 describes the fields of the TMR_FIPER register.

**Table 14-128. TMR_FIPER Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–31 | FIPER | Fixed interval pulse period register. This field must be programmed to an integer multiple of TMR_CTRL[TCLK_PERIOD] value to ensure a period pulse being generated correctly. |

### 14.5.3.11.14 External Trigger Stamp Register (TMR_ETTS1–2_H/L)

General purpose external trigger -stamp register (TMR_ETTS*n*_H/L). This register holds time at the programmable edge of the external trigger. The registers in eTSEC1 are shared for all eTSECs. This register is read only in normal operation. Figure 14-121 describes the definition for the TMR_ETTS*n*_H/L register.

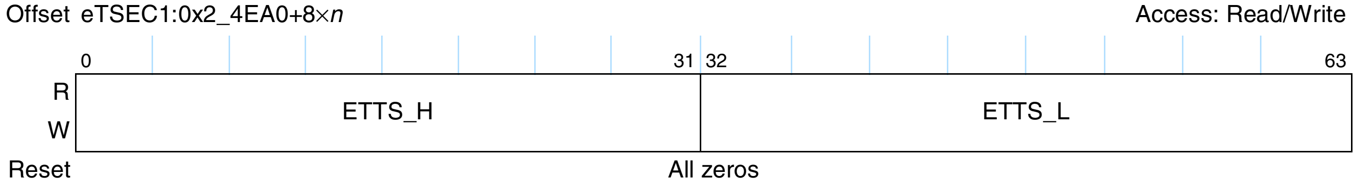


**Figure 14-121. TMR_ETTS1-2_H/L Register Definition**

Table 14-129 describes the fields of the TMR_ETTS*n*_H/L register.

**Table 14-129. TMR_ETTS1-2_H Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–63 | ETTS_H/L | Time stamp field at the programmable edge of the external trigger. |

## 14.5.4 Ten-Bit Interface (TBI)

This section describes the ten-bit interface (TBI), reduced ten-bit interface (RTBI), and the TBI/RTBI MII set of registers. TBI and RTBI operate in the same manner (the only difference is that RTBI has reduced I/O signalling).

### 14.5.4.1 TBI Transmit Process

The eTSEC's TBI implements the transmit portion of the physical coding sublayer as found in Clause 36 of IEEE 802.3z. In SerDes mode, packets conveyed across the GMII are encapsulated and encoded into 10-bit symbols and output to the SerDes. In GMII mode, the GMII signals are passed through to the attached GMII PHY.

#### 14.5.4.1.1 Packet Encapsulation

If TX_EN is de-asserted the eTSEC outputs an idle stream. If TX_EN is asserted, a Start_of_Packet symbol is output. This symbol replaces the first byte of the preamble field. All other bytes of the packet pass through an 8B10B encoding module. After the last byte of the FCS field is signaled through the GMII, the MAC de-asserts TX_EN. The eTSEC then outputs an End_of_Packet symbol. Then, depending on the position of the End_of_Packet symbols (being in either an odd or even position) the eTSEC outputs one or two Carrier_Extend symbols. Following the last Carrier_Extend symbol, the eTSEC resumes sending idle codes. If, during a packet, the eTSEC wishes to mark a byte invalid, TX_ER is asserted. The eTSEC, upon detection of TX_ER, substitutes the data symbol for an Error_Propagation symbol.

#### 14.5.4.1.2 8B10B Encoding

Every eight-bit data octet has two (not necessarily different) ten-bit symbols associated with it. Depending on the running disparity (the cumulative difference of ones and zeroes) the eTSEC module chooses the appropriate symbol.

Special encapsulation symbols are called ordered_sets. Ordered_sets are comprised of one to four ten-bit symbols. Ordered_sets can be found in Clause 36 of the IEEE 802.3z specification.

#### 14.5.4.1.3 Preamble Shortening

Because the idle ordered_set comprises two symbols and begins on an even symbols boundary, packets can only begin on an even boundary. However, the GMII has no such restriction and may signal TX_EN on an odd boundary. If this happens, the eTSEC delays the Start_of_Packet symbol, effectively ignoring the first byte of preamble; thus, a seven octet preamble becomes six octets on the Ten-Bit Interface.

### 14.5.4.2 TBI Receive Process

The eTSEC's TBI Implements the receive portion of the physical coding sublayer as found in Clause 36 of IEEE 802.3z. The Receive portion includes the Synchronization state machine. In SerDes mode, the eTSEC first attempts to acquire synchronization on the link by examining received symbols. Once synchronization is acquired, received packets are decoded and sent across the Receive GMII interface. In GMII mode, the GMII signals are passed through to the MAC.

#### 14.5.4.2.1 Synchronization

The eTSEC examines received symbols looking for the seven bit 'comma' string embedded in some special symbols. Both the idle ordered_set and the Configuration ordered_set contain a symbol which has the comma. Once a certain number of codes with comma are detected, the eTSEC is considered to have acquired synchronization.

#### 14.5.4.2.2 Auto-Negotiation for 1000BASE-X

Once synchronization is acquired, ordered_sets are decoded. If Configuration ordered_sets are received, the eTSEC decodes the two octet data field and the sixteen-bit Configuration data is stored and used to Auto-Negotiate with the link partner. in the Receive Configuration Register (RXCR[15:0]) an internal register used to receive all the link partners informations and used to compare to local ability during negotiation. Not visible to user. If, during Auto-Negotiation an invalid symbol is detected, Auto-Negotiation re-starts. After Auto-Negotiation is completed the TBI MII Status Register SR[AN done] in set. In this mode, packets may be received from the link partner.

### 14.5.4.3 TBI MII Set Register Descriptions

This section describes the TBI MII registers. All of the TBI registers are 16 bits wide. The TBI registers are accessed at the offset of the TBI physical address. The eTSEC's TBI physical address is stored in the TBIPA register. Writing to the TBI registers is performed in a way similar to writing to an external PHY, using the MII management interface. Using TBIPA in place of the PHY address, in the MIIMADD[PHY Address] field, and setting the MIIMADD[Register Address] to the appropriate address offset that corresponds to the register that one wants to read or write (see Table 14-130), the user can read (set MIIMCOM[read cycle]) or write (writing to MIIMCON[PHY control]) to the TBI block. Refer to the TBI physical address register in Section 14.5.3.1, "eTSEC General Control and Status Registers," and the TBI MII register set in Table 14-130. Notice that jitter diagnostics and TBI control are not IEEE 802.3 required registers and are only used for test and control of the eTSEC TBI block. The TBI's TBI control register (TBI) is for configuring the eTSEC ten-bit interface block. However, because this TBI block has an MII management interface (just like any other PHY), it has an IEEE 802.3 register called the control register (CR).

**Table 14-130. TBI MII Register Set**

| Offset Address | Name | Access | Size | Section/page |
|---|---|---|---|---|
| TEN-BIT INTERFACE (TBI) REGISTERS | | | | 14.5.4/14-133 |
| 0x00 | Control (CR) | R/W[1] | 16 bits | 14.5.4.3.1/14-136 |
| 0x01 | Status (SR) | R, LH, LL | 16 bits | 14.5.4.3.2/14-137 |
| 0x02–0x03 | Reserved | R | 2 bytes | — |
| 0x04 | AN advertisement (ANA) | RW, R | 16 bits | 14.5.4.3.2/14-137 |
| 0x05 | AN link partner base page ability (ANLPBPA) | R | 16 bits | 14.5.4.3.4/14-140 |
| 0x06 | AN expansion (ANEX) | R, LH | 16 bits | 14.5.4.3.5/14-141 |
| 0x07 | AN next page transmit (ANNPT) | R/W, R | 16 bits | 14.5.4.3.6/14-141 |
| 0x08 | AN link partner ability next page (ANLPANP) | R | 16 bits | 14.5.4.3.7/14-142 |
| 0x0F | Extended status (EXST) | R | 16 bits | 14.5.4.3.8/14-143 |
| 0x10 | Jitter diagnostics (JD) | R/W | 16 bits | 14.5.4.3.9/14-144 |
| 0x11 | TBI control (TBICON) | R/W | 16 bits | 14.5.4.3.10/14-145 |

[1] R = Read-only, WO = Write Only, R/W = Read and Write, LH = Latches High, LL = Latches Low, SC = Self-clearing,

### 14.5.4.3.1 Control Register (CR)

Figure 14-122 describes the definition for the CR register.

Offset 0x00 Access: Read/Write

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | | | | | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | PHY Reset | — | Speed[0] | AN Enable | — | | Reset AN | Full Duplex | — | Speed[1] | — | | | | | |
| Reset | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

**Figure 14-122. Control Register Definition**

Table 14-131 describes the fields of the CR register.

**Table 14-131. CR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | PHY Reset | PHY reset. This bit is cleared by default. This bit is self-clearing.<br>0 Normal operation.<br>1 The internal state of the TBI is reset. This in turn may change the state of the TBI link partner. |
| 1 | — | Reserved |
| 2 | Speed[0] | Speed selection. This bit defaults to a cleared state and should always be cleared, which corresponds to 1000 Mbps speed.Setting this field controls the speed at which the TBI operates. The table for Speed[1] provides the appropriate encoding. Its default is bit[2] = ‘0’; bit[9] = ’1’. |
| 3 | AN Enable | Auto-negotiation enable. This bit is set by default.<br>0 The values programmed in bits 2, 7 and 9 determine the operating condition of the link.<br>1 Auto-negotiation process enabled. |
| 4–5 | — | Reserved |
| 6 | Reset AN | Reset auto-negotiation. This bit is cleared by default and is self-clearing.<br>0 Normal operation.<br>1 The auto-negotiation process restarts. This action is only available if auto-negotiation is enabled. |
| 7 | Full Duplex | Duplex mode. This bit is set by default.<br>0 Reserved.<br>1 Full-duplex operation. |
| 8 | — | Reserved, should be cleared. |
| 9 | Speed[1] | Speed selection. This bit defaults to a set state and should always be set, which corresponds to 1000 Mbps speed.Setting this field controls the speed at which the TBI operates. The following table provides the appropriate encoding. Its default is bit[2] = ‘0’; bit[9] = ’1’. |
| 10–15 | — | Reserved |

| Maximum Operating Speed | Bit 2 | Bit 9 |
|---|---|---|
| Reserved | 0 | 0 |
| Reserved | 1 | 0 |
| 1000 Mbps | 0 | 1 |
| Reserved | 1 | 1 |

#### 14.5.4.3.2 Status Register (SR)

Figure 14-123 describes the definition for the SR register.

Offset 0x01

Access: Read only

| | 0 | | | | | | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | — | | | | | | | Extend Status | — | No Pre | AN Done | Remote Fault | AN Ability | Link Status | — | Extend Ability |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |

**Figure 14-123. Status Register Definition**

Table 14-132 describes the fields of the SR register.

**Table 14-132. SR Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–6 | — | Reserved, should be cleared. |
| 7 | Extend Status | This bit indicates that PHY status information is also contained in the Register 15, Extended Status Register. Returns 1 on read. This bit is read-only. |
| 8 | — | Reserved, should be cleared. |
| 9 | No Pre | MF preamble suppression enable. This bit indicates whether or not the PHY is capable of handling MII management frames without the 32-bit preamble field. Returns 1, indicating support for suppressed preamble MII management frames. This bit is read-only. |
| 10 | AN Done | Auto-negotiation complete. This bit is read-only and is cleared by default.<br>0 Either the auto-negotiation process is underway or the auto-negotiation function is disabled.<br>1 The auto-negotiation process has completed. |
| 11 | Remote Fault | Remote fault. This bit is read-only and is cleared by default. Each read of the status register clears this bit.<br>0 Normal operation.<br>1 A remote fault condition was detected. This bit latches high in order for software to detect the condition. |
| 12 | AN Ability | Auto-negotiation ability. While read as set, this bit indicates that the PHY has the ability to perform auto-negotiation. While read as cleared, this bit indicates the PHY lacks the ability to perform auto-negotiation. Returns 1 on read. This bit is read-only. |
| 13 | Link Status | Link status. This bit is read-only and is cleared by default.<br>0 A valid link is not established. This bit latches low allowing for software polling to detect a failure condition.<br>1 A valid link is established. |
| 14 | — | Reserved, should be cleared. |
| 15 | Extend Ability | Extended capability. This bit indicates that the PHY contains the extended set of registers (those beyond control and status). Returns 1 on read. This bit is read-only. |

### 14.5.4.3.3 AN Advertisement Register (ANA)

Figure 14-124 describes the definition for the ANA register.

Offset 0x04 Access: Read/Write

| | 0 | 1 | 2 3 | 4 6 | 7 8 | 9 | 10 | 11 15 |
|---|---|---|---|---|---|---|---|---|
| R<br>W | Next Page | — | Remote Fault | — | Pause | Half Duplex | Full Duplex | — |
| Reset | All zeros | | | | | | | |

**Figure 14-124. AN Advertisement Register Definition**

Table 14-133 describes the fields of the ANA register.

**Table 14-133. ANA Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | Next Page | Next page configuration. The local device sets this bit to either request next page transmission or advertise next page exchange capability.<br>0 The local device wishes not to engage in next page exchange.<br>1 The local device has no next pages but wishes to allow reception of next pages. If the local device has no next pages and the link partner wishes to send next pages, the local device shall send null message codes and have the message page set to 0b000_0000_0001, as defined in annex 28C. |
| 1 | — | Reserved. (Ignore on read) |
| 2–3 | Remote Fault | The local device's remote fault condition is encoded in bits 2 and 3 of the base page. Values are shown in the following table. The default value is 00. Indicate a fault by setting a non-zero remote fault encoding and re-negotiating.<br>RF1 bit[3] \| RF2 bit[2] \| Description<br>0 \| 0 \| No error, link OK<br>0 \| 1 \| Offline<br>1 \| 0 \| Link_Failure<br>1 \| 1 \| Auto-Negotiation_Error |
| 4–6 | — | Reserved, should be cleared. |
| 7–8 | Pause | The local device's PAUSE capability is encoded in bits 7 and 8, and the decodes are shown in the following table. For priority resolution information consult Table 14-134.<br>PAUSE bit[8] \| ASM_DIR bit[7] \| Capability<br>0 \| 0 \| No PAUSE<br>0 \| 1 \| Asymmetric PAUSE toward link partner<br>1 \| 0 \| Symmetric PAUSE<br>1 \| 1 \| Both symmetric PAUSE and Asymmetric PAUSE toward local device |

**Table 14-133. ANA Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 9 | Half Duplex | Half-duplex capability.<br>0 Designates local device as not capable of half-duplex operation.<br>1 Designates local device as capable of half-duplex operation. |
| 10 | Full Duplex | Full-duplex capability.<br>0 Designates the local device as not capable of full-duplex operation.<br>1 Designates the local device as capable of full-duplex operation. |
| 11–15 | — | Reserved, should be cleared. |

Table 14-134 describes the resolution of pause priority.

**Table 14-134. PAUSE Priority Resolution**

| Local Device | | Link Partner | | Local Resolution | Link Partner Resolution |
|---|---|---|---|---|---|
| PAUSE | ASM_DIR | PAUSE | ASM_DIR | | |
| 0 | 0 | x | x | Disable PAUSE transmit<br>Disable PAUSE receive | Disable PAUSE transmit<br>Disable PAUSE receive |
| 0 | 1 | 0 | x | Disable PAUSE transmit<br>Disable PAUSE receive | Disable PAUSE transmit<br>Disable PAUSE receive |
| 0 | 1 | 1 | 0 | Disable PAUSE transmit<br>Disable PAUSE receive | Disable PAUSE transmit<br>Disable PAUSE receive |
| 0 | 1 | 1 | 1 | Enable PAUSE transmit<br>Disable PAUSE receive | Disable PAUSE transmit<br>Enable PAUSE receive |
| 1 | 0 | 0 | x | Disable PAUSE transmit<br>Disable PAUSE receive | Disable PAUSE transmit<br>Disable PAUSE receive |
| 1 | 0 | 1 | x | Enable PAUSE transmit<br>Enable PAUSE receive | Enable PAUSE transmit<br>Enable PAUSE receive |
| 1 | 1 | 0 | 0 | Disable PAUSE transmit<br>Disable PAUSE receive | Disable PAUSE transmit<br>Disable PAUSE receive |
| 1 | 1 | 0 | 1 | Disable PAUSE transmit<br>Enable PAUSE receive | Enable PAUSE transmit<br>Disable PAUSE receive |
| 1 | 1 | 1 | x | Enable PAUSE transmit<br>Enable PAUSE receive | Enable PAUSE transmit<br>Enable PAUSE receive |

#### 14.5.4.3.4 AN Link Partner Base Page Ability Register (ANLPBPA)

Figure 14-125 describes the definition for the ANLPBPA register.

Offset 0x05 | Access: Read/Write

| | 0 | 1 | 2 3 | 4 6 | 7 8 | 9 | 10 | 11 15 |
|---|---|---|---|---|---|---|---|---|
| R | Next Page | — | Remote Fault | — | Pause | Half Duplex | Full Duplex | — |
| W | | | | | | | | |
| Reset | All zeros | | | | | | | |

**Figure 14-125. AN Link Partner Base Page Ability Register Definition**

Table 14-135 describes the fields of the ANLPBPA register.

**Table 14-135. ANLPBPA Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | Next Page | Next page. This bit is read-only. The link partner sets or clears this bit.<br>0 Link partner has no subsequent next pages or is not capable of receiving next pages.<br>1 Link partner either requesting next page transmission or indicating the capability to receive next pages. |
| 1 | — | Reserved. (Ignore on read) |
| 2–3 | Remote Fault | The link partner's remote fault condition is encoded in bits 2 and 3 of the base page. Values are shown in the remote fault encoding field table below. This bit is read-only. |
| 4–6 | — | Reserved, should be cleared. |
| 7–8 | Pause | Encoding of the link partner's PAUSE capability is shown in the PAUSE encoding table below. For priority resolution information consult. This bit is read-only |
| 9 | Half Duplex | Half-duplex capability. This bit is read-only.<br>0 Link partner is not capable of half-duplex mode.<br>1 Link partner is capable of half-duplex mode. |

| RF1 bit[3] | RF2 bit[2] | Description |
|---|---|---|
| 0 | 0 | No error, link OK |
| 0 | 1 | Offline |
| 1 | 0 | Link_Failure |
| 1 | 1 | Auto-Negotiation_Error |

| PAUSE bit[8] | ASM_DIR bit[7] | Capability |
|---|---|---|
| 0 | 0 | No PAUSE |
| 0 | 1 | Asymmetric PAUSE toward link partner |
| 1 | 0 | Symmetric PAUSE |
| 1 | 1 | Both symmetric PAUSE and<br>Asymmetric PAUSE toward local device |

**Table 14-135. ANLPBPA Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 10 | Full Duplex | Full-duplex capability. This bit is read-only.<br>0 Link partner is not capable of full-duplex mode.<br>1 Link partner is capable of full-duplex mode. |
| 11–15 | — | Reserved, should be cleared. |

### 14.5.4.3.5 AN Expansion Register (ANEX)

Figure 14-126 describes the definition for the ANEX register.

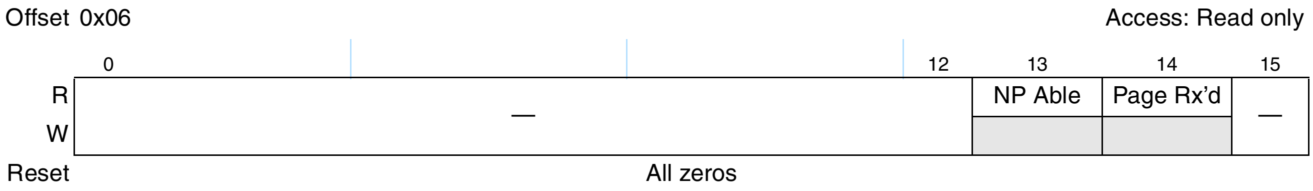


**Figure 14-126. AN Expansion Register Definition**

Table 14-136 describes the fields of the ANEX register.

**Table 14-136. ANEX Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–12 | — | Reserved, should be cleared. |
| 13 | NP Able | Next page able. This bit is read-only and returns 1 on read. While read as set, indicates local device supports next page function. |
| 14 | Page Rx'd | Page received. This bit is read-only. The bit clears on a read to the register.<br>0 Normal operation.<br>1 A new page was received and stored in the applicable AN link partner ability or AN next page register. This bit latches high in order for software to detect while polling. |
| 15 | — | Reserved, should be cleared. |

### 14.5.4.3.6 AN Next Page Transmit Register (ANNPT)

Figure 14-127 describes the definition for the ANNPT register.

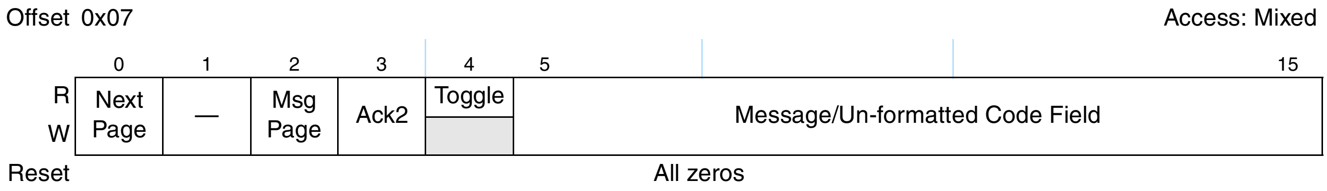


**Figure 14-127. AN Next Page Transmit Register Definition**

Table 14-137 describes the fields of the ANNPT register.

**Table 14-137. ANNPT Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | Next Page | Next page indication. [Reference MII bit 7.15 in IEEE 802.3, 2000 Edition Clause 28.2.4]<br>0 Last page.<br>1 Additional next pages to follow. |
| 1 | — | Reserved. (Ignore on read) |
| 2 | Msg Page | Message page. [Reference MII bit 7.13]<br>0 Unformatted page.<br>1 Message page. |
| 3 | Ack2 | Acknowledge 2. Used by the next page function to indicate that the device has the ability to comply with the message. [Reference MII bit 7.12]<br>0 The local device cannot comply with message.<br>1 The local device complies with message. |
| 4 | Toggle | Toggle. Used to ensure synchronization with the link partner during next page exchange. This bit always takes the opposite value of the toggle bit of the previously-exchanged link code word. The initial value in the first next page transmitted is the inverse of bit 11 in the base link code word. [Reference MII bit 7.11] This bit is read-only.<br>0 Toggle bit of the previously-exchanged link code word was 1.<br>1 Toggle bit of the previously-exchanged link code word was 0. |
| 5–15 | Message/ Un-formatted Code Field | Message pages are formatted pages that carry a pre-defined message code, which is enumerated in IEEE 802.3u/Annex 28C. Unformatted code fields take on an arbitrary value. [Reference MII field 7.10:0] |

### 14.5.4.3.7 AN Link Partner Ability Next Page Register (ANLPANP)

Figure 14-128 describes the definition for the ANLPANP register.

Offset 0x08 Access: Read only

| | 0 | 1 | 2 | 3 | 4 | 5 … 15 |
|---|---|---|---|---|---|---|
| R | Next Page | — | Msg Page | Ack2 | Toggle | Message/Un-formatted Code Field |
| W | | | | | | |
| Reset | All zeros | | | | | |

**Figure 14-128. AN Link Partner Ability Next Page Register Definition**

Table 14-138 describes the fields of the ANLPANP register.

**Table 14-138. ANLPANP Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | Next Page | Next page. The link partner sets and clears this bit.<br>0 Last page from link partner<br>1 Additional next pages to follow |
| 1 | — | Reserved. (Ignore on read) |
| 2 | Msg Page | Message page.<br>0 Unformatted page<br>1 Message page |

**Table 14-138. ANLPANP Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 3 | Ack2 | Acknowledge 2. Indicates the link partner's ability to comply with the message.<br>0 Link partner cannot comply with message.<br>1 Link partner complies with message. |
| 4 | Toggle | Toggle. Used to ensure synchronization with the link partner during next page exchange. This bit always takes the opposite value of the toggle bit of the previously-exchanged link code word. The initial value in the first next page transmitted is the inverse of bit 11 in the base link code word. This bit is read-only.<br>0 Toggle bit of the previously-exchanged link code word was 1.<br>1 Toggle bit of the previously-exchanged link code word was 0. |
| 5–15 | Message/ Un-formatted Code Field | Message pages are formatted pages that carry a pre-defined message code, which is enumerated in IEEE 802.3u/Annex 28C. Unformatted code fields take on an arbitrary value. |

### 14.5.4.3.8 Extended Status Register (EXST)

Figure 14-129 describes the definition for the EXST register.

Offset 0x0F — Access: Read only

| | 0 | 1 | 2 | 3 | 4 | | | | | | | | | | | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | 1000X Full | 1000X Half | 1000T Full | 1000T Half | — | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Figure 14-129. Extended Status Register Definition**

Table 14-139 describes the fields of the EXST register.

**Table 14-139. EXST Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | 1000X Full | 1000X full-duplex capability. Returns 1 on read. This bit is read-only.<br>0 PHY cannot operation in 1000BASE-X full-duplex mode.<br>1 PHY can operate in 1000BASE-X full-duplex mode. |
| 1 | 1000X Half | 1000X half-duplex capability. Returns 0 on read. This bit is read-only.<br>0 PHY cannot operation in 1000BASE-X half-duplex mode.<br>1 PHY can operate in 1000BASE-X half-duplex mode. |
| 2 | 1000T Full | 1000T full-duplex capability. Returns 1 on read. This bit is read-only.<br>0 PHY cannot operation in 1000BASE-T full-duplex mode.<br>1 PHY can operate in 1000BASE-T full-duplex mode. |
| 3 | 1000T Half | 1000T half-duplex capability. Returns 0 on read. This bit is read-only.<br>0 PHY cannot operation in 1000BASE-T half-duplex mode.<br>1 PHY can operate in 1000BASE-T half-duplex mode. |
| 4–15 | — | Reserved |

### 14.5.4.3.9 Jitter Diagnostics Register (JD)

Annex 36A in IEEE 802.3z describes several jitter test patterns. These can be configured to be sent by writing the jitter diagnostics register. See the register description for more information. In may be wise to auto-negotiate and advertise a remote fault signaling of offline prior to beginning the test patterns. Figure 14-130 describes the definition for the JD register.

<table>
<tr><td>Offset 0x10</td><td colspan="4"></td><td>Access: Read/Write</td></tr>
<tr><td></td><td>0</td><td>1 3</td><td>4 5</td><td>6 15</td><td></td></tr>
<tr><td>R<br>W</td><td>Jitter Enable</td><td>Jitter Select</td><td>—</td><td>Custom Jitter Pattern</td><td></td></tr>
<tr><td>Reset</td><td colspan="4">All zeros</td><td></td></tr>
</table>

**Figure 14-130. Jitter Diagnostics Register Definition**

Table 14-140 describes the fields of the JD register.

**Table 14-140. JD Field Descriptions**

<table>
<tr><th>Bits</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Jitter Enable</td><td>Jitter enable. This bit is cleared by default.<br>0 Normal transmit operation.<br>1 Enable the TBI to transmit the jitter test patterns defined in IEEE 802.3z 36A.</td></tr>
<tr><td>1–3</td><td>Jitter Select</td><td>Selects the jitter pattern to be transmitted in diagnostics mode. Encoding of this field is shown in the following table. Default is 00.
<table>
<tr><th>Jitter Pattern Select</th><th>bit[1]</th><th>bit[2]</th><th>bit[3]</th></tr>
<tr><td>User defined uses custom jitter pattern, bits 6–15</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>High frequency (+/- D21.5)<br>1010101010101010101010101010101010101010…</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td>Mixed frequency (+/- K28.5)<br>1111101011000001010011111010110000010100…</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>Low frequency<br>1111100000111110000011111000001111100000…</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>Complex pattern (10’h17c,10’h0c9,10’h0e5,10’h2a3, 10’h17c,...)</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>Square Wave (- K28.7)<br>0011111000001111100000111110000011111000…</td><td>1</td><td>0</td><td>1</td></tr>
<tr><td>Reserved</td><td>1</td><td>1</td><td>0</td></tr>
<tr><td>Reserved</td><td>1</td><td>1</td><td>1</td></tr>
</table>
</td></tr>
<tr><td>4–5</td><td>—</td><td>Reserved</td></tr>
<tr><td>6–15</td><td>Custom Jitter Pattern</td><td>Used in conjunction with jitter (pattern) select and jitter (diagnostic) enable; set this field to the desired custom pattern which is continuously transmitted. Its default is 0x000.</td></tr>
</table>

#### 14.5.4.3.10 TBI Control Register (TBICON)

Figure 14-131 describes the definition for the TBICON register.

Offset 0x11 Access: Mixed

|  | 0 | 1 | 2 | 3 | 4 |  | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Soft_Reset | — | Disable Rx Dis | Disable Tx Dis | — |  |  | AN Sense | — |  | Clock Select | MII Mode | — |  |  |  |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

**Figure 14-131. TBI Control Register Definition**

Table 14-141 describes the fields of the TBICON register.

**Table 14-141. TBICON Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | Soft_Reset | Soft reset. This bit is cleared by default.<br>0 Normal operation.<br>1 Resets the functional modules in the TBI. |
| 1 | — | Reserved. (Ignore on read) |
| 2 | Disable Rx Dis | Disable receive disparity. This bit is cleared by default.<br>0 Normal operation.<br>1 Disables the running disparity calculation and checking in the receive direction. |
| 3 | Disable Tx Dis | Disable transmit disparity. This bit is cleared by default.<br>0 Normal operation.<br>1 Disables the running disparity calculation and checking in the transmit direction. |
| 4–6 | — | Reserved |
| 7 | AN Sense | Auto-negotiation sense enable. This bit is cleared by default.<br>0 IEEE 802.3z Clause 37 behavior is desired, which results in the link not completing.<br>1 Allow the auto-negotiation function to sense either a Gigabit MAC in auto-negotiation bypass mode or an older Gigabit MAC without auto-negotiation capability. If sensed, auto-negotiation complete becomes true; however, the page received is low, indicating no page was exchanged. Management can then act accordingly. |
| 8–9 | — | Reserved |
| 10 | Clock Select | Clock select. This bit is cleared by default.<br>0 Allow the TBI to accept dual split-phase 62.5 MHz receive clocks.<br>1 Configure the TBI to accept a 125 MHz receive clock from the SerDes/PHY. The 125 MHz clock must be physically connected to 'PMA receive clock 0' if using a parallel (non-SGMII) Ethernet protocol. |
| 11 | MI Mode | This bit describes the configuration mode of the TBI. The user reads a 1 while the TBI is configured in GMII/MII mode (connected to a GMII/MII PHY) and a 0 while configured in TBI mode (connected to a 1000BASE-X SerDes). Its value is the inverse of ECNTRL[TBIM].<br>0 TBI mode.<br>1 GMII mode. |
| 12–15 | — | Reserved |

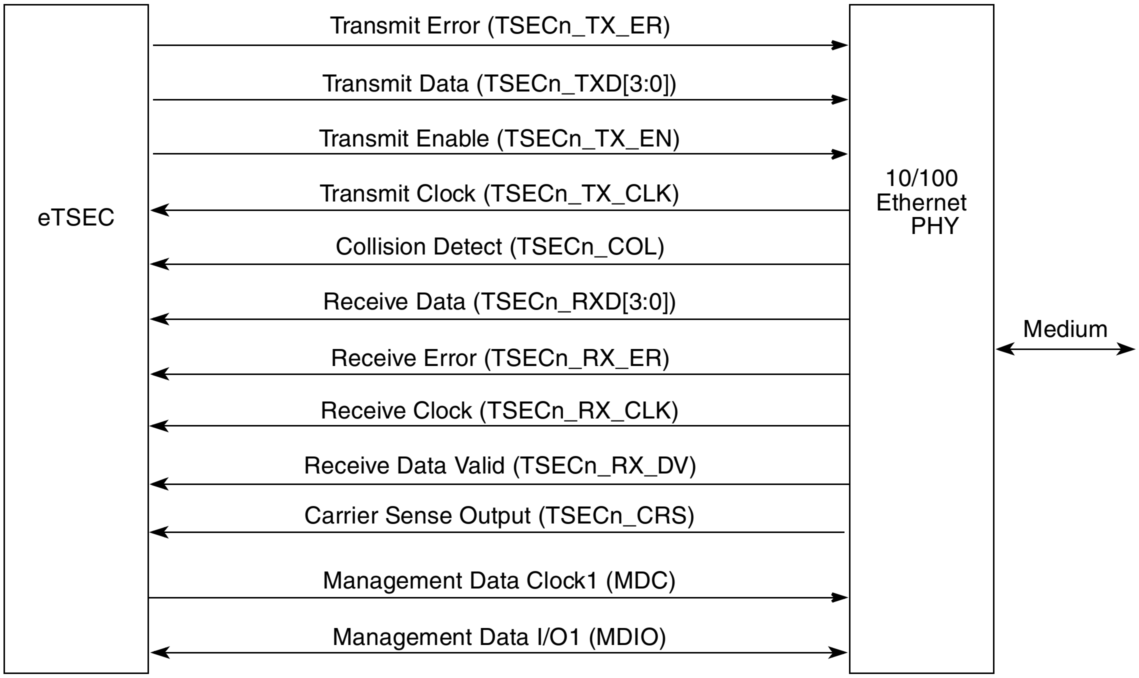
## 14.6 Functional Description

### 14.6.1 Connecting to Physical Interfaces on Ethernet

This section describes how to connect the eTSEC to various interfaces: MII, GMII, RMII, RGMII, TBI, and RTBI. To avoid confusion, all of the buses follow the bus conventions used in the IEEE 802.3 specification because the PHYs follow the same conventions. (For instance, in the bus TSEC*n*_TXD[7:0], bit 7 is the msb and bit 0 is the lsb). If a mode does not use all input signals available to a particular eTSEC, those inputs that are not used must be pulled low on the board.

#### 14.6.1.1 Media-Independent Interface (MII)

This section describes the media-independent interface (MII) intended to be used between the PHYs and the eTSEC. Figure 14-132 depicts the basic components of the MII including the signals required to establish eTSEC module connection with a PHY.


eTSEC
Transmit Error (TSECn_TX_ER)
Transmit Data (TSECn_TXD[3:0])
Transmit Enable (TSECn_TX_EN)
Transmit Clock (TSECn_TX_CLK)
Collision Detect (TSECn_COL)
Receive Data (TSECn_RXD[3:0])
Receive Error (TSECn_RX_ER)
Receive Clock (TSECn_RX_CLK)
Receive Data Valid (TSECn_RX_DV)
Carrier Sense Output (TSECn_CRS)
Management Data Clock1 (MDC)
Management Data I/O1 (MDIO)
10/100 Ethernet PHY
Medium

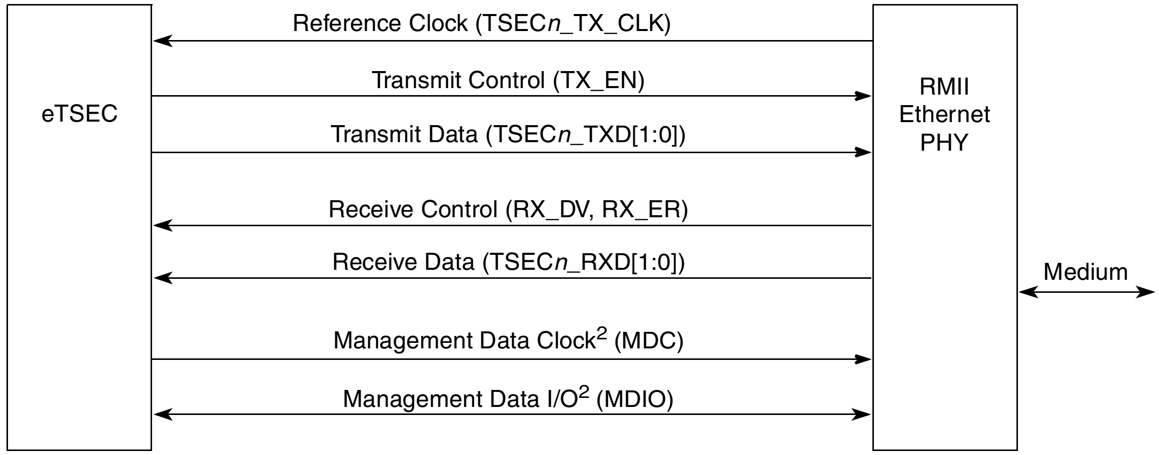
[1] The management signals (MDC and MDIO) are common to all of the Ethernet controllers' connections in the system, assuming that each PHY has a different management address.

**Figure 14-132. eTSEC-MII Connection**

An MII interface has 18 signals (including the MDC and MDIO signals), as defined by the IEEE 802.3u standard, for connecting to an Ethernet PHY.

#### 14.6.1.2 Reduced Media-Independent Interface (RMII)

This section describes the reduced media-independent interface (RMII) intended to be used between the PHYs and the GMII MAC. The RMII is a reduced-pin alternative to the IEEE 802.3u MII. The RMII reduces the number of signals required to interconnect the MAC and the PHY from a maximum of 18 signals (MII) to 10 signals. To accomplish this objective, the data paths are halved in width and clocked at twice the MII clock frequency, while clocks, carrier sense and error signals have been partly combined. For 100 Mbps operation, the reference clock operates at 50 MHz, whereas for 10 Mbps operation, the clock remains at 50MHz, but only every 10th cycle is used. Figure 14-133 depicts the basic components of the reduced media-independent interface and the signals required to establish an eTSEC's connection with a PHY. The RMII is implemented as defined by the RMII Specification of the RMII Consortium, as of March 20, 1998.

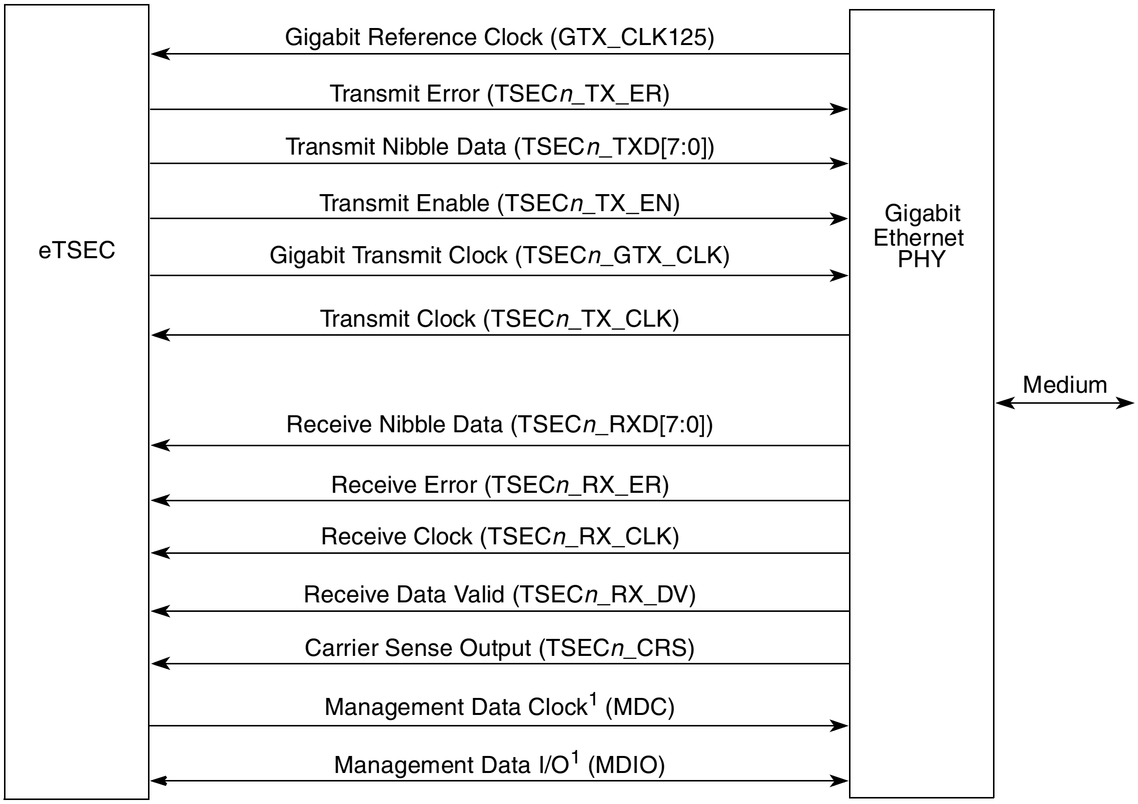

eTSEC
Reference Clock (TSECn_TX_CLK)
Transmit Control (TX_EN)
Transmit Data (TSECn_TXD[1:0])
Receive Control (RX_DV, RX_ER)
Receive Data (TSECn_RXD[1:0])
Management Data Clock² (MDC)
Management Data I/O² (MDIO)
RMII Ethernet PHY
Medium

[2] The management signals (MDC and MDIO) are common to all of the Ethernet controllers module connections in the system, assuming that each PHY has a different management address.

**Figure 14-133. eTSEC-RMII Connection**

### 14.6.1.3 Gigabit Media-Independent Interface (GMII)

This section describes the gigabit media-independent interface (GMII) intended to be used between the PHYs and the eTSEC. Figure 14-134 depicts the basic components of the GMII including the signals required to establish the eTSEC module connection with a PHY.


eTSEC
Gigabit Ethernet PHY
Gigabit Reference Clock (GTX_CLK125)
Transmit Error (TSECn_TX_ER)
Transmit Nibble Data (TSECn_TXD[7:0])
Transmit Enable (TSECn_TX_EN)
Gigabit Transmit Clock (TSECn_GTX_CLK)
Transmit Clock (TSECn_TX_CLK)
Receive Nibble Data (TSECn_RXD[7:0])
Receive Error (TSECn_RX_ER)
Receive Clock (TSECn_RX_CLK)
Receive Data Valid (TSECn_RX_DV)
Carrier Sense Output (TSECn_CRS)
Management Data Clock1 (MDC)
Management Data I/O1 (MDIO)
Medium

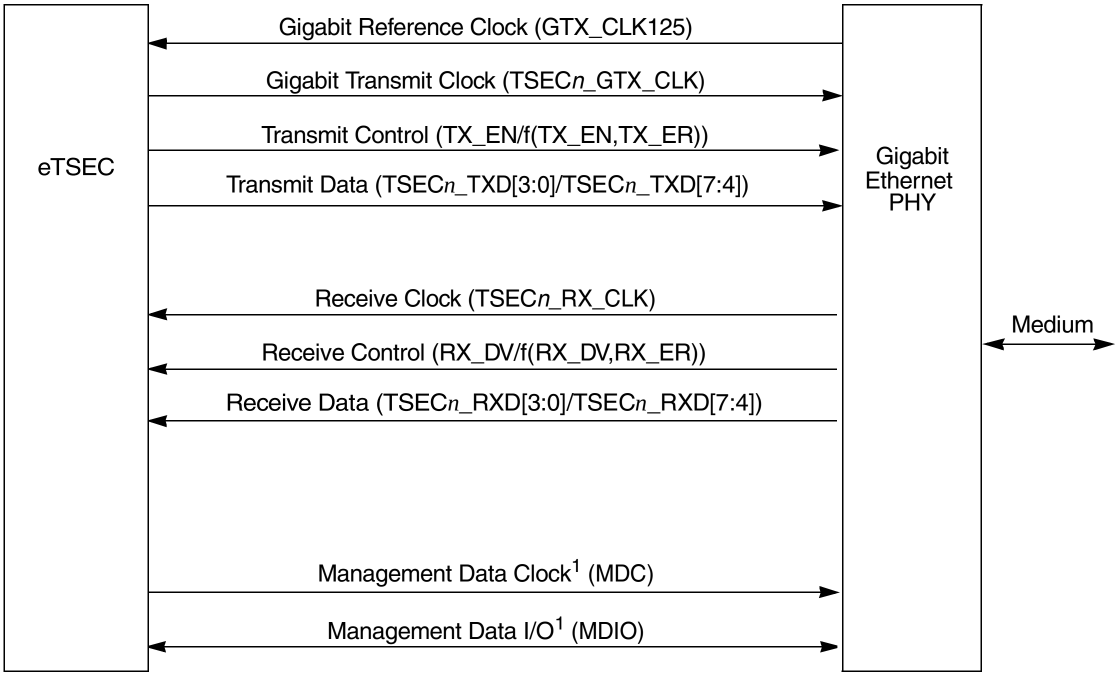
[1] The management signals (MDC and MDIO) are common to all of the Ethernet controllers' connections in the system, assuming that each PHY has a different management address.

**Figure 14-134. eTSEC-GMII Connection**

A GMII interface has 28 signals (TSEC*n*_GTX_CLK + EC_GTX_CLK125 included), as defined by the IEEE 802.3u standard, for connecting to an Ethernet PHY.

### 14.6.1.4 Reduced Gigabit Media-Independent Interface (RGMII)

This section describes the reduced gigabit media-independent interface (RGMII) intended to be used between the PHYs and the GMII MAC. The RGMII is an alternative to the IEEE802.3u MII, the IEEE802.3z GMII and the TBI. The RGMII reduces the number of signals required to interconnect the MAC and the PHY from a maximum of 28 signals (GMII) to 15 signals (GTX_CLK125 included) in a cost effective and technology independent manner. To accomplish this objective, the data paths and all associated control signals are multiplexed using both edges of the clock. For gigabit operation, the clocks operate at 125MHz, and for 10/100 operation, the clocks operate at 2.5 MHz or 25 MHz, respectively. Note that the GTX_CLK125 input must be provided at 125 MHz for an RGMII interface, regardless of operation speed (1 Gbps, 100 Mbps, or 10 Mbps). Figure 14-135 depicts the basic components of the gigabit reduced media-independent interface and the signals required to establish the gigabit Ethernet controllers' module connection with a PHY. The RGMII is implemented as defined by the RGMII specification Version 1.2a 9/22/00.

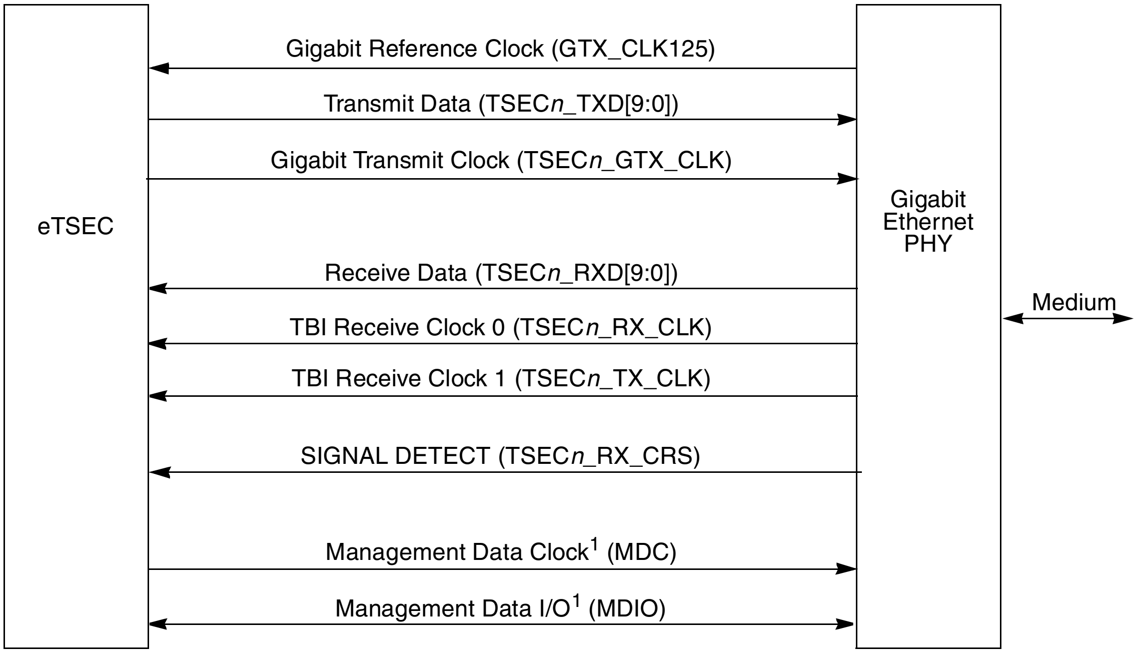

eTSEC
Gigabit Reference Clock (GTX_CLK125)
Gigabit Transmit Clock (TSECn_GTX_CLK)
Transmit Control (TX_EN/f(TX_EN,TX_ER))
Transmit Data (TSECn_TXD[3:0]/TSECn_TXD[7:4])
Receive Clock (TSECn_RX_CLK)
Receive Control (RX_DV/f(RX_DV,RX_ER))
Receive Data (TSECn_RXD[3:0]/TSECn_RXD[7:4])
Management Data Clock1 (MDC)
Management Data I/O1 (MDIO)
Gigabit Ethernet PHY
Medium

[1] The management signals (MDC and MDIO) are common to all of the gigabit Ethernet controllers module connections in the system, assuming that each PHY has a different management address.

**Figure 14-135. eTSEC-RGMII Connection**

### 14.6.1.5 Ten-Bit Interface (TBI)

This section describes the ten-bit interface (TBI) intended to be used between the PHYs and the eTSEC to implement a standard SerDes interface for optical-fiber devices in 1000BASE-SX/LX applications. Figure 14-136 depicts the basic components of the TBI including the signals required to establish eTSEC module connection with a PHY. RBC0 and RBC1 are differential 62.5 MHz receive clocks. If not connected to the TBI PHY, the Signal Detect (SDET) input must be tied high. This causes the eTSEC to begin auto negotiation with the SERDES immediately upon the TBI module being enabled.


eTSEC
Gigabit Reference Clock (GTX_CLK125)
Transmit Data (TSECn_TXD[9:0])
Gigabit Transmit Clock (TSECn_GTX_CLK)
Receive Data (TSECn_RXD[9:0])
TBI Receive Clock 0 (TSECn_RX_CLK)
TBI Receive Clock 1 (TSECn_TX_CLK)
SIGNAL DETECT (TSECn_RX_CRS)
Management Data Clock1 (MDC)
Management Data I/O1 (MDIO)
Gigabit Ethernet PHY
Medium

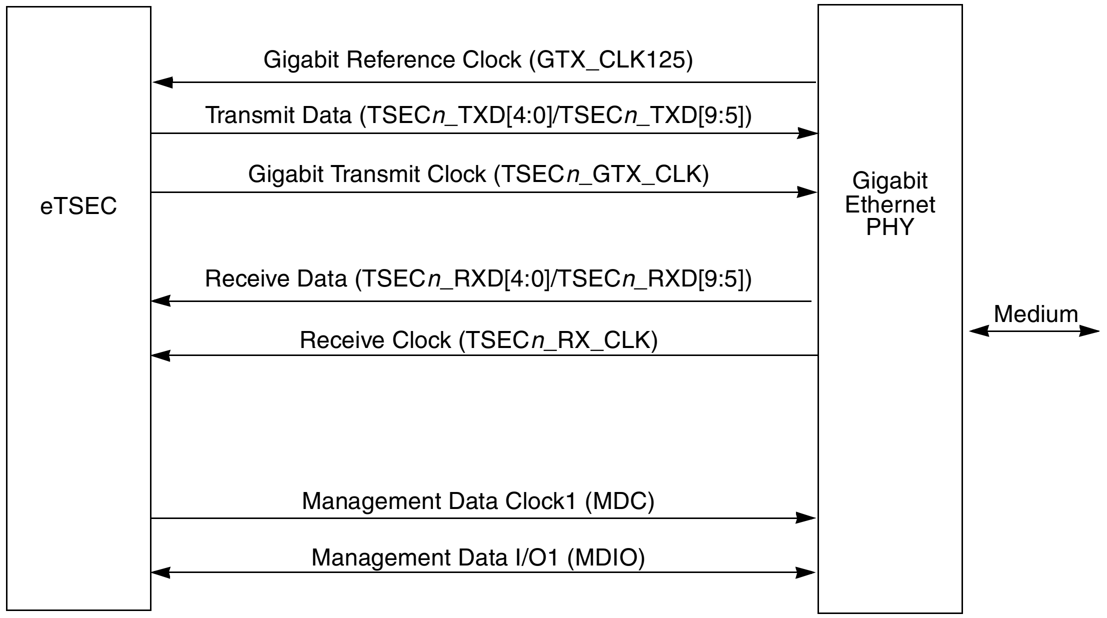
[1] The management signals (MDC and MDIO) are common to all of the Ethernet controllers' connections in the system, assuming that each PHY has a different management address.

**Figure 14-136. eTSEC-TBI Connection**

A TBI interface has 26 signals (GE_GTX_CLK125 included) for connecting to an Ethernet PHY, as defined by IEEE 802.3z GMII and TBI standards.

### 14.6.1.6 Reduced Ten-Bit Interface (RTBI)

This section describes the reduced ten-bit interface (RTBI) intended to be used between the PHYs and the eTSEC to implement a reduced-pin count version of a SerDes interface for optical-fiber devices in 1000BASE-SX/LX applications. Figure 14-137 depicts the basic components of the RTBI including the signals required to establish eTSEC module connection with a PHY. Note that in RTBI the eTSEC immediately begins auto-negotiation with the SerDes.


eTSEC
Gigabit Reference Clock (GTX_CLK125)
Transmit Data (TSECn_TXD[4:0]/TSECn_TXD[9:5])
Gigabit Transmit Clock (TSECn_GTX_CLK)
Receive Data (TSECn_RXD[4:0]/TSECn_RXD[9:5])
Receive Clock (TSECn_RX_CLK)
Management Data Clock1 (MDC)
Management Data I/O1 (MDIO)
Gigabit Ethernet PHY
Medium

[1] The management signals (MDC and MDIO) are common to all of the Ethernet controllers' connections in the system, assuming that each PHY has a different management address.

**Figure 14-137. eTSEC-RTBI Connection**

A RTBI interface has 15 signals (GE_GTX_CLK125 included), as defined by the RGMII specification Version 1.2a 9/22/00, and is intended to be an alternative to the IEEE 802.3u MII, the IEEE 802.3z GMII and the TBI standard for connecting to an Ethernet PHY.

### 14.6.1.7 Ethernet Physical Interfaces Signal Summary

Table 14-142 describes the signal multiplexing for the following interfaces: GMII, MII, TBI, and RMII.

**Table 14-142. GMII, MII, and RMII Signals Multiplexing**

| eTSEC Signals | | | GMII Interface | | | MII Interface | | | RMII Interface | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Frequency [MHz] 125 | | | Frequency [MHz] 125 | | | Frequency [MHz] 25 | | | Frequency [MHz] 50 | | |
| Voltage[V] 3.3/2.5 | | | Voltage[V] 3.3 | | | Voltage[V] 3.3 | | | Voltage[V] 3.3 | | |
| Signals (TSEC*n*_) | I/O | No. of Signals | Signals (TSEC*n*_) | I/O | No. of Signals | Signals (TSEC*n*_) | I/O | No. of Signals | Signals (TSEC*n*_) | I/O | No. of Signals |
| GTX_CLK | O | 1 | GTX_CLK | O | 1 | | | | | | |
| TX_CLK | I | 1 | TX_CLK | I | 1 | TX_CLK | I | 1 | REF_CLK[1] | I | 1 |
| TxD[0] | O | 1 | TxD[0] | O | 1 | TxD[0] | O | 1 | TxD[0] | O | 1 |
| TxD[1] | O | 1 | TxD[1] | O | 1 | TxD[1] | O | 1 | TxD[1] | O | 1 |
| TxD[2] | O | 1 | TxD[2] | O | 1 | TxD[2] | O | 1 | | | |
| TxD[3] | O | 1 | TxD[3] | O | 1 | TxD[3] | O | 1 | | | |
| TxD[4] | O | 1 | TxD[4] | O | 1 | | | | | | |
| TxD[5] | O | 1 | TxD[5] | O | 1 | | | | | | |
| TxD[6] | O | 1 | TxD[6] | O | 1 | | | | | | |
| TxD[7] | O | 1 | TxD[7] | O | 1 | | | | | | |
| TX_EN | O | 1 | TX_EN | O | 1 | TX_EN | O | 1 | TX_EN | O | 1 |
| TX_ER | O | 1 | TX_ER | O | 1 | TX_ER | O | 1 | | | |
| RX_CLK | I | 1 | RX_CLK | I | 1 | RX_CLK | I | 1 | | | |
| RxD[0] | I | 1 | RxD[0] | I | 1 | RxD[0] | I | 1 | RxD[0] | I | 1 |
| RxD[1] | I | 1 | RxD[1] | I | 1 | RxD[1] | I | 1 | RxD[1] | I | 1 |
| RxD[2] | I | 1 | RxD[2] | I | 1 | RxD[2] | I | 1 | | | |
| RxD[3] | I | 1 | RxD[3] | I | 1 | RxD[3] | I | 1 | | | |
| RxD[4] | I | 1 | RxD[4] | I | 1 | | | | | | |
| RxD[5] | I | 1 | RxD[5] | I | 1 | | | | | | |
| RxD[6] | I | 1 | RxD[6] | I | 1 | | | | | | |
| RxD[7] | I | 1 | RxD[7] | I | 1 | | | | | | |
| RX_DV | I | 1 | RX_DV | I | 1 | RX_DV | I | 1 | CRS_DV | I | 1 |
| RX_ER | I | 1 | RX_ER | I | 1 | RX_ER | I | 1 | RX_ER | I | 1 |
| COL | I | 1 | | | | COL | I | 1 | | | |
| CRS | I | 1 | | | | CRS | I | 1 | | | |
| **Sum** | | 25 | **Sum** | | 23 | **Sum** | | 16 | **Sum** | | 8 |

[1]

Table 14-143 describes the signal multiplexing for RGMII, TBI, and RTBI interfaces.

**Table 14-143. RGMII, TBI, and RTBI Signals Multiplexing**

| **eTSEC Signals** | | | **RGMII Interface** | | | **TBI Interface** | | | **RTBI Interface** | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Frequency [MHz] 125** | | | **Frequency [MHz] 125** | | | **Frequency [MHz] 62.5** | | | **Frequency [MHz] 62.5** | | |
| **Voltage[V] 3.3/2.5** | | | **Voltage[V] 2.5** | | | **Voltage[V] 3.3** | | | **Voltage[V] 2.5** | | |
| **Signals (TSEC*n*_)** | **I/O** | **No. of Signals** | **Signals (TSEC*n*_)** | **I/O** | **No. of Signals** | **Signals (TSEC*n*_)** | **I/O** | **No. of Signals** | **Signals (TSEC*n*_)** | **I/O** | **No. of Signals** |
| GTX_CLK | O | 1 | GTX_CLK | O | 1 | GTX_CLK | O | 1 | GTX_CLK | O | 1 |
| TX_CLK | I | 1 | | | | RX_CLK1 | I | 1 | | | |
| TxD[0] | O | 1 | TxD[0]/TxD[4] | O | 1 | TCG[0] | O | 1 | TCG[0]/TCG[5] | O | 1 |
| TxD[1] | O | 1 | TxD[1]/TxD[5] | O | 1 | TCG[1] | O | 1 | TCG[1]/TCG[6] | O | 1 |
| TxD[2] | O | 1 | TxD[2]/TxD[6] | O | 1 | TCG[2] | O | 1 | TCG[2]/TCG[7] | O | 1 |
| TxD[3] | O | 1 | TxD[3]/TxD[7] | O | 1 | TCG[3] | O | 1 | TCG[3]/TCG[8] | O | 1 |
| TxD[4] | O | 1 | | | | TCG[4] | O | 1 | | | |
| TxD[5] | O | 1 | | | | TCG[5] | O | 1 | | | |
| TxD[6] | O | 1 | | | | TCG[6] | O | 1 | | | |
| TxD[7] | O | 1 | | | | TCG[7] | O | 1 | | | |
| TX_EN | O | 1 | TX_CTL (TX_EN/ TX_ERR) | O | 1 | TCG[8] | O | 1 | TCG[4]/TCG[9] | O | 1 |
| TX_ER | O | 1 | | | | TCG[9] | O | 1 | | | |
| RX_CLK | I | 1 | RX_CLK | I | 1 | RX_CLK0 | I | 1 | RX_CLK | I | 1 |
| RxD[0] | I | 1 | RxD[0]/RxD[4] | I | 1 | RCG[0] | I | 1 | RCG[0]/RCG[5] | I | 1 |
| RxD[1] | I | 1 | RxD[1]/RxD[5] | I | 1 | RCG[1] | I | 1 | RCG[1]/RCG[6] | I | 1 |
| RxD[2] | I | 1 | RxD[2]/RxD[6] | I | 1 | RCG[2] | I | 1 | RCG[2]/RCG[7] | I | 1 |
| RxD[3] | I | 1 | RxD[3]/RxD[7] | I | 1 | RCG[3] | I | 1 | RCG[3]/RCG[8] | I | 1 |
| RxD[4] | I | 1 | | | | RCG[4] | I | 1 | | | |
| RxD[5] | I | 1 | | | | RCG[5] | I | 1 | | | |
| RxD[6] | I | 1 | | | | RCG[6] | I | 1 | | | |
| RxD[7] | I | 1 | | | | RCG[7] | I | 1 | | | |
| RX_DV | I | 1 | RX_CTL (RX_DV/ RX_ERR) | I | 1 | RCG[8] | I | 1 | RCG[4]/RCG[9] | I | 1 |
| RX_ER | I | 1 | | | | RCG[9] | I | 1 | | | |
| COL | I | 1 | | | | | | | | | |
| CRS | I | 1 | | | | SDET | I | 1 | | I | |
| **Sum** | | 25 | **Sum** | | 12 | **Sum** | | 24 | **Sum** | | 12 |

Table 14-144 describes the signal multiplexing for RGMII and RTBI interfaces.

Table 14-144. RGMII and RTBI Signals Multiplexing

| eTSEC Signals | | | RGMII Interface | | | RTBI Interface | | |
|---|---|---|---|---|---|---|---|---|
| Frequency [MHz] 125 | | | Frequency [MHz] 125 | | | Frequency [MHz] 62.5 | | |
| Voltage[V] 3.3/2.5 | | | Voltage[V] 2.5 | | | Voltage[V] 2.5 | | |
| **Signals (TSEC*n*_)** | **I/O** | **No. of Signals** | **Signals (TSEC*n*_)** | **I/O** | **No. of Signals** | **Signals (TSEC*n*_)** | **I/O** | **No. of Signals** |
| GTX_CLK | O | 1 | GTX_CLK | O | 1 | GTX_CLK | O | 1 |
| TX_CLK | I | 1 | | | | | | |
| TxD[0] | O | 1 | TxD[0]/TxD[4] | O | 1 | TCG[0]/TCG[5] | O | 1 |
| TxD[1] | O | 1 | TxD[1]/TxD[5] | O | 1 | TCG[1]/TCG[6] | O | 1 |
| TxD[2] | O | 1 | TxD[2]/TxD[6] | O | 1 | TCG[2]/TCG[7] | O | 1 |
| TxD[3] | O | 1 | TxD[3]/TxD[7] | O | 1 | TCG[3]/TCG[8] | O | 1 |
| TxD[4] | O | 1 | | | | | | |
| TxD[5] | O | 1 | | | | | | |
| TxD[6] | O | 1 | | | | | | |
| TxD[7] | O | 1 | | | | | | |
| TX_EN | O | 1 | TX_CTL (TX_EN/ TX_ERR) | O | 1 | TCG[4]/TCG[9] | O | 1 |
| TX_ER | O | 1 | | | | | | |
| RX_CLK | I | 1 | RX_CLK | I | 1 | RX_CLK | I | 1 |
| RxD[0] | I | 1 | RxD[0]/RxD[4] | I | 1 | RCG[0]/RCG[5] | I | 1 |
| RxD[1] | I | 1 | RxD[1]/RxD[5] | I | 1 | RCG[1]/RCG[6] | I | 1 |
| RxD[2] | I | 1 | RxD[2]/RxD[6] | I | 1 | RCG[2]/RCG[7] | I | 1 |
| RxD[3] | I | 1 | RxD[3]/RxD[7] | I | 1 | RCG[3]/RCG[8] | I | 1 |
| RxD[4] | I | 1 | | | | | | |
| RxD[5] | I | 1 | | | | | | |
| RxD[6] | I | 1 | | | | | | |
| RxD[7] | I | 1 | | | | | | |
| RX_DV | I | 1 | RX_CTL (RX_DV/ RX_ERR) | I | 1 | RCG[4]/RCG[9] | I | 1 |
| RX_ER | I | 1 | | | | | | |
| COL | I | 1 | | | | | | |
| CRS | I | 1 | | | | | I | |
| **Sum** | | 25 | **Sum** | | 12 | **Sum** | | 12 |

**Table 14-145. RGMII Signals Multiplexing**

| **eTSEC Signals** | | | **RGMII Interface** | | |
|---|---|---|---|---|---|
| **Frequency [MHz] 125** | | | **Frequency [MHz] 125** | | |
| **Voltage[V] 3.3/2.5** | | | **Voltage[V] 2.5** | | |
| **Signals (TSEC*n*_)** | **I/O** | **No. of Signals** | **Signals (TSEC*n*_)** | **I/O** | **No. of Signals** |
| GTX_CLK | O | 1 | GTX_CLK | O | 1 |
| TX_CLK | I | 1 | | | |
| TxD[0] | O | 1 | TxD[0]/TxD[4] | O | 1 |
| TxD[1] | O | 1 | TxD[1]/TxD[5] | O | 1 |
| TxD[2] | O | 1 | TxD[2]/TxD[6] | O | 1 |
| TxD[3] | O | 1 | TxD[3]/TxD[7] | O | 1 |
| TxD[4] | O | 1 | | | |
| TxD[5] | O | 1 | | | |
| TxD[6] | O | 1 | | | |
| TxD[7] | O | 1 | | | |
| TX_EN | O | 1 | TX_CTL (TX_EN/TX_ERR) | O | 1 |
| TX_ER | O | 1 | | | |
| RX_CLK | I | 1 | RX_CLK | I | 1 |
| RxD[0] | I | 1 | RxD[0]/RxD[4] | I | 1 |
| RxD[1] | I | 1 | RxD[1]/RxD[5] | I | 1 |
| RxD[2] | I | 1 | RxD[2]/RxD[6] | I | 1 |
| RxD[3] | I | 1 | RxD[3]/RxD[7] | I | 1 |
| RxD[4] | I | 1 | | | |
| RxD[5] | I | 1 | | | |
| RxD[6] | I | 1 | | | |
| RxD[7] | I | 1 | | | |
| RX_DV | I | 1 | RX_CTL (RX_DV/RX_ERR) | I | 1 |
| RX_ER | I | 1 | | | |
| COL | I | 1 | | | |
| CRS | I | 1 | | | |
| **Sum** | | 25 | **Sum** | | 12 |

Table 14-146 describes the signals shared by all interfaces.

Table 14-146. Shared Signals

| Signals | I/O | No. of Signals | Function |
|---|---|---|---|
| MDIO | I/O | 1 | Management interface I/O |
| MDC | O | 1 | Management interface clock |
| GTX_CLK125 | I | 1 | Reference clock |
| **Sum** | | 3 | — |

#### 14.6.1.8 SGMII Interface

SGMII communication using the eTSEC is accomplished through the SerDes interface. See Table 14-1 on page 14-7 for specific signal assignments.

### 14.6.2 Connecting to FIFO Interfaces

This section describes how to connect an eTSEC to third-party communication devices, including users' ASICs and FPGAs, through the FIFO interface.

Each eTSEC provides an 8-bit full-duplex packet FIFO interface port that bypasses the Ethernet MAC, but re-uses the GMII signals. As a result, the FIFO interface normally does not impose the overheads of Ethernet framing. The FIFO interface operates synchronously, at a maximum frequency defined by a ratio of 4.2:1 (platform:TxClk) in GMII mode and 3.2:1 (platform:TxClk) in encoded mode providing OC-48 full-duplex transfer rates. For example, a FIFO frequency of 127 MHz in GMII mode requires a platform frequency of 533 MHz; a FIFO frequency of 200 MHz in encoded mode requires a platform frequency of 667 MHz; a FIFO frequency of 167 MHz in encoded mode requires a platform frequency of 533 MHz.

Bare IP packets—with an optional 32-bit CRC check sequence—can be transferred to the eTSEC directly. The eTSEC Tx and Rx FIFOs, TOE functions, and DMA continue to be used in packet FIFO mode.

The ECNTRL[FIFM] bit determines whether eTSEC is communicating with its Ethernet MAC or FIFO interface.

- 8-bit packet FIFO
  - The GMII signals of each eTSEC can be used to create a FIFO port, therefore eTSEC can support up to two simultaneous 8-bit FIFO interfaces Choosing between 8-bit FIFO and Ethernet affects each eTSEC independently, therefore a mix of FIFO and Ethernet interfaces can be configured.
  - The data signals of GMII and 8-bit FIFO remain the same. The data valid (RX_DV, TX_EN) and error (RX_ER, TX_ER) signals are used to signal framing information. If required, the collision (COL) and carrier sense (CRS) signals can be used in an encoded mode to provide link-level flow control.

The following restrictions apply in any of the FIFO modes:

- Transferred packets must be no more than 9600 bytes in length.

- If RCTRL[PRSFM]=0, received packets must be a minimum of 10 bytes.
- If RCTRL[PRSFM]=1, received packets must be a minimum of 14 bytes.
- Transmitted packets with L2 headers must be a minimum of 14 bytes.
- Transmitted packets without L2 headers must be a minimum of 10 bytes.
- Although TCP/IP offload is supported, the receive queue filer table must be limited to as many entries as eTSEC can search every packet. See Section 14.6.5.2.1, “Filing Rules,” on page 14-184 for guidance on how to determine maximum table size for an application.
- eTSEC requires received packets to have a minimum inter-packet gap of three cycles.
- On transmission, the minimum inter-packet gap (set in FIFOCFG[IPG]) is three cycles if CRC is not automatically appended. Each CRC data beat adds to this requirement. For 8-bit FIFO interfaces the minimum is 7 cycles.

No Ethernet-specific features (such as MAC address matching) or layer 2 properties (such as Ethertype) are available in FIFO mode.

#### 14.6.2.1 Flow Control

In the encoded (non GMII-style) FIFO modes, link-level flow control is provided to the eTSEC transmitter on the COL signal of the controlling eTSEC, while back pressure to the remote transmitter is sent on the CRS signal (which acts as an output signal only in FIFO mode). Owing to the synchronization delay of responding to flow control on signal COL, the eTSEC cannot stop transmission immediately, but may require up to 8 clock cycles before transmission is paused. The eTSEC issues flow control either when software forces it (through the FIFOCFG[FFC] bit), or when the Rx FIFO reaches its high watermark.

#### 14.6.2.2 CRC Appending and Checking

If FIFOCFG[CRCAPP] is enabled, the FIFO interface automatically appends a 4-byte CRC to each transmitted packet. Alternatively, if FIFOCFG[CRCAPP] is cleared, TxBD[TC] provides a per-packet override to append CRC. The IEEE 802.3 standard CRC-32 algorithm is used, where the least significant bit of each byte (TXD[0]) is combined into the CRC ahead of the most significant bit (TXD[]). Accordingly, the CRC result, CRC[31:0] is transmitted onto the interface in bit-reversed order, CRC[24:31], CRC[16:23], CRC[8:15], CRC[0:7].

Automatic checking of CRC-32 checksums received over the FIFO interface is enabled by setting FIFOCFG[CRCCHK]. CRC errors are recorded in the RxBD[CR] flag of every last buffer. Like transmit, the receiver combines data into the CRC in the order least significant data bit (RXD[0]) to most significant bit (RXD[]). The last 4 bytes of the packet are assumed to be CRC whenever FIFOCFG[CRCCHK] is enabled, and these bytes are returned as part of the data buffer.

### 14.6.2.3 8-Bit GMII-Style Packet FIFO Mode

Figure 14-138 depicts the signals required to establish eTSEC module connection with an external device using the 8-bit FIFO interface.

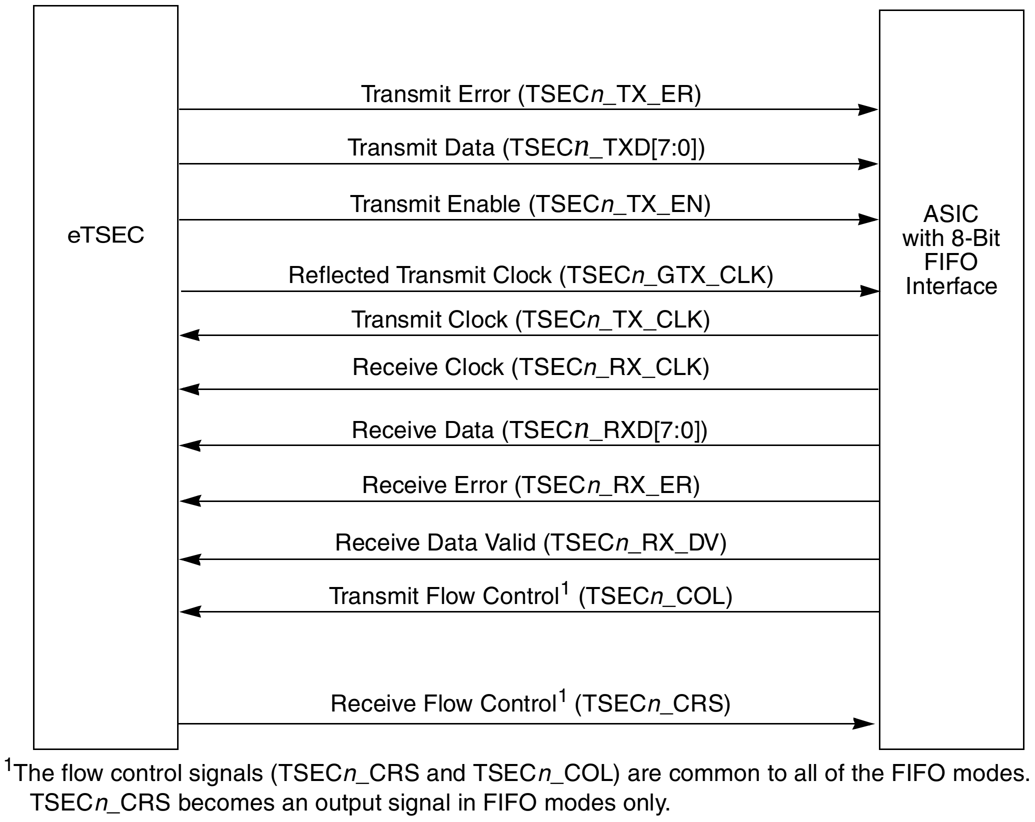

eTSEC
ASIC with 8-Bit FIFO Interface
Transmit Error (TSECn_TX_ER)
Transmit Data (TSECn_TXD[7:0])
Transmit Enable (TSECn_TX_EN)
Reflected Transmit Clock (TSECn_GTX_CLK)
Transmit Clock (TSECn_TX_CLK)
Receive Clock (TSECn_RX_CLK)
Receive Data (TSECn_RXD[7:0])
Receive Error (TSECn_RX_ER)
Receive Data Valid (TSECn_RX_DV)
Transmit Flow Control1 (TSECn_COL)
Receive Flow Control1 (TSECn_CRS)
1The flow control signals (TSECn_CRS and TSECn_COL) are common to all of the FIFO modes.
TSECn_CRS becomes an output signal in FIFO modes only.

**Figure 14-138. eTSEC-FIFO (8-Bit) Connection**

The 8-bit FIFO interface has 25 signals (including the flow control signals). Illustrative timing of the GMII-style FIFO mode is shown in Figure 14-139.

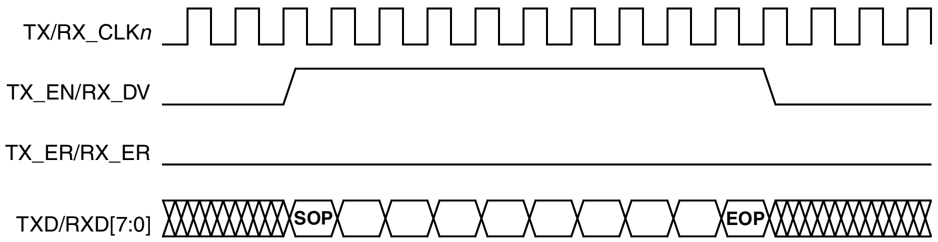


**Figure 14-139. 8-Bit GMII-Style Packet FIFO Timing**

The encoding of the eTSEC GMII signals in this FIFO mode is shown in Table 14-147.

Table 14-147. Signal Encoding for GMII-Style 8-Bit FIFO

| Condition | TX_EN/RX_DV | TX_ER/RX_ER |
|---|---|---|
| Valid data, start of packet | 0 to 1 transition at start of cycle | 0 |
| Valid data | 1 | 0 |
| Valid data, end of packet | 1 to 0 transition at end of cycle | 0 |
| Error | 1 | 1 until TX_EN/RX_DV falls |

In this mode flow control can control only the decision to continue transmitting packets, as packet transfers cannot be suspended once started.

### 14.6.2.4 8-Bit Encoded Packet FIFO Mode

The encoded packet 8-bit FIFO mode uses the signals shown in Figure 14-140. The control lines encode four states that can be associated with each beat of data. This mode should be used where invalid bytes can appear between the start and end of packet. Illustrative timing of the encoded packet FIFO mode is shown in Figure 14-140.

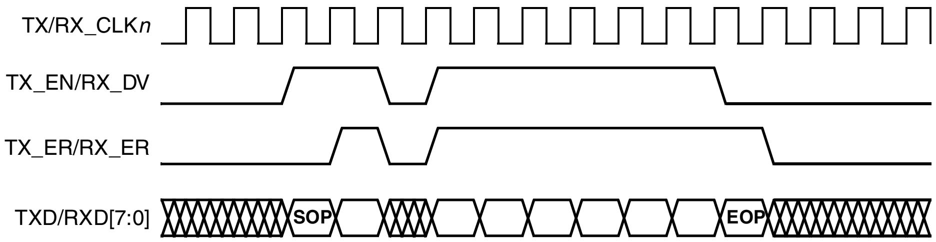


Figure 14-140. 8-Bit Encoded Packet FIFO Timing

The encoding of the eTSEC GMII signals in this FIFO mode is shown in Table 14-148.

Table 14-148. Signal Encoding for Encoded 8-Bit FIFO

| Condition | TX_EN/RX_DV | TX_ER/RX_ER |
|---|---|---|
| Valid data, start of packet | 1 | 0 |
| Valid data | 1 | 1 |
| Valid data, end of packet | 0 | 1 |
| Data not valid | 0 | 0 |

In this mode flow control can cause an indefinite number of invalid data bytes to be transferred. This is the only mode in which an empty eTSEC Tx FIFO also causes a string of invalid data bytes to be transmitted rather than causing an underrun error.

### 14.6.2.5 FIFO Interface Signal Summary

Refer to Section 14.7.1.7, “8-Bit FIFO Mode” for interface signal details.

## 14.6.3 Gigabit Ethernet Controller Channel Operation

This section describes the operation of the eTSEC. First, the software initialization sequence is described. Next, the software (Ethernet driver) interface for transmitting and receiving frames is reviewed. Frame filtering and receive filing algorithm features are also discussed. The section concludes with interrupt handling, inter-packet gap time, and loop back descriptions.

### 14.6.3.1 Initialization Sequence

This sections describes which registers are reset due to a hard or software reset and what registers the user must initialize prior to enabling the eTSEC.

#### 14.6.3.1.1 Hardware Controlled Initialization

A hard reset occurs when the system powers up. All eTSEC's registers and control logic are reset to their default states after a hard reset has occurred. In this state, each eTSEC behaves like a PowerQUICC III device, except for the absence of out-of-sequence TxBD features. That is, initially TCP/IP off-load is disabled and only single RxBD and TxBD rings are accessible.

#### 14.6.3.1.2 User Initialization

After the system has undergone a hard reset, software must initialize certain basic eTSEC registers. Other registers can also be initialized during this time, but they are optional and must be determined based on the requirements of the system. See Table 14-3 for the register list. Table 14-149 describes the minimum steps for register initialization.

**Table 14-149. Steps for Minimum Register Initialization**

| Description |
|---|
| 1. Set and clear MACCFG1 [Soft_Reset] |
| 2. Initialize MACCFG2 |
| 3. Initialize MAC station address |
| 4. Set up the PHY using the MII Mgmt Interface |
| 5. Configure the TBI control to TBI or GMII |
| 6. Clear IEVENT |
| 7. Initialize IMASK |
| 8. Initialize RCTRL |
| 9. Initialize DMACTRL |

After the initialization of registers is performed, the user must execute the following steps in the order described below to bring the eTSEC into a functional state (out of reset):

1. Write to the MACCFG1 register and set the appropriate bits. These need to include RX_EN and TX_EN. To enable flow control, Rx_Flow and Tx_Flow should also be set.

2. For the transmission of Ethernet frames, TxBDs must first be built in memory, linked together as a ring, and pointed to by the TBASE*n* registers. A minimum of two buffer descriptors per ring is required, unless the ring is disabled. Setting the ring to a size of one causes the same frame to be transmitted twice. If TCP/IP off-load is to be enabled, the TxBD[TOE] bit must be set for each frame.
3. Likewise, for the reception of Ethernet frames, the receive queue (or queues) must be ready, with its RxBD pointed to by the RBASE*n* registers. If TCP/IP off-load is to be enabled, RCTRL[PRSDEP] must be set to the required off-load level. Both transmit and receive can be gracefully stopped after transmission and reception begins.
4. Clearing DMACTRL[GTS] triggers the transmission of frame data if the transmitter had been previously stopped. The DMACTRL[GRS] must be cleared if the receiver had been previously stopped. Refer to the DMACTRL register section, and Section 14.6.8.1, “Data Buffer Descriptors,” for more information.

### 14.6.3.2 Soft Reset and Reconfiguring Procedure

Before issuing a soft-reset to and/or reconfiguring the MAC with new parameters, the user must properly shutdown the DMA and make sure it is in an idle state for the entire duration. User must gracefully stop the DMA by setting both GRS and GTS bits in the DMACTRL register, then wait for both GRSC and GTSC bits to be set in the IEVENT register before resetting the MAC or changing parameters. Both GRS and GTS bits must be cleared before re-enabling the MAC to resume the DMA.

During the MAC configuration, if a new set of Tx buffer descriptors are used, the user must load the pointers into the TBASE registers. Likewise if a new set of Rx buffer descriptors are used, the RBASE registers must be written with new pointers.

Following is a procedure to gracefully reset and reconfigure the MAC:

1. Set GRS/GTS bits in DMACTRL register
2. Poll GRSC/GTSC bits in IEVENT register until both are set
3. Set SOFT_RESET bit in MACCFG1 register (Note that SOFT_RESET must remain set for at least 3 TX clocks before proceeding.)
4. Clear SOFT_RESET bit in MACCFG1 register
5. Load TDBPH, TBASEH, TBASE0–TBASE7 with new Tx BD pointers
6. Load RDBPH, RBASEH, RBASE0–RBASE7 with new Rx BD pointers
7. Setup other MAC registers (MACCFG2, MAXFRM, and so on)
8. Setup group address hash table (GADDR0–GADDR15) if address filtering is required
9. Setup receive frame filer table (through RQFAR, RQFCR, and RQFPR) if filing to multiple RxBD rings is required
10. Setup WWR, WOP, TOD bits in DMACTRL register
11. Enable transmit queues in TQUEUE, and ensure that the transmit scheduling mode is correctly set in TCTRL.
12. Enable receive queues in RQUEUE, and optionally set TOE functionality in RCTRL.
13. Clear THLT and TXF bits in TSTAT register by writing 1 to them

14. Clear QHLT and RXF bits in RSTAT register by writing 1 to them.
15. Clear GRS/GTS bits in DMACTRL (do not change other bits)
16. Enable Tx_EN/Rx_EN in MACCFG1 register

### 14.6.3.3 Gigabit Ethernet Frame Transmission

The Ethernet transmitter requires little core intervention. After the software driver initializes the system, the eTSEC begins to poll the first transmit buffer descriptor (TxBD) in TxBD ring 0 every 512 transmit clocks. If TxBD[R] is set, and the TxBD ring is scheduled for transmission, the eTSEC begins copying the associated transmit buffer from memory to its Tx FIFO. The transmitter takes data from the Tx FIFO and transmits data to the MAC. The MAC transmits the data through the GMII interface to the physical media. The transmitter, once initialized, runs until the end-of-frame (EOF) condition is detected unless a collision within the collision window occurs (half-duplex mode) or an abort condition is encountered.

If the user has a frame ready to transmit, setting the DMACTRL[TOD] eliminates waiting for the next poll and a DMA transfer of the transmit data buffers can begin immediately. The transmission begins once all data for the frame is loaded into the Tx FIFO or sufficient transmit data (determined by the Tx FIFO threshold register) is in the Tx FIFO. If the line is not busy, the MAC transmit logic asserts TX_EN and sends the 7-octet preamble sequence, 1-octet start of frame delimiter, and frame information in that order. If the line is busy, the controller waits for the carrier sense signal, CRS, to remain inactive for 60 bit times (60 clocks) and transmission begins after an additional 36 bit times (96 bit times after CRS became active). In full-duplex mode, because collisions are ignored, frame transmission maintains only the interframe gap (96 bit times) regardless of CRS.

In half-duplex mode (MACCFG2[Full Duplex] is cleared) the MAC defers transmission if the line is busy (CRS asserted). Before transmitting, the MAC waits for carrier sense to become inactive, at which point it then determines if CRS remains negated for 60 clocks. If so, transmission begins after an additional 36 bit times (96 bit times after CRS originally became negated). If CRS continues to be asserted, the MAC follows a specified back-off procedure and tries to retransmit the frame until the retry limit is reached. Data stored in the Tx FIFO is re-transmitted in case of a collision. This avoids unnecessary memory traffic.

The transmitter also monitors for an abort condition and terminates the current frame if an abort condition is encountered. In full-duplex mode the protocol is independent of network activity, and only the transmit inter-frame gap must be enforced.

The transmitter implements full-duplex flow control. If a flow control frame is received, the MAC does not service the transmitter’s request to send data until the pause duration is over. If the MAC is currently sending data after a pause frame has been received and processed, the MAC finishes sending the current frame, then suspends subsequent frames (except a pause frame) until the pause duration is over. In addition, the transmitter supports transmission of flow control frames through TCTRL[TFC_PAUSE]. The transmit pause frame is generated internally based on the PAUSE register that defines the pause value to be sent. Note that it is possible to send a pause frame while the pause timer has not expired.

The MAC automatically appends FCS (32-bit CRC) bytes to the frame if any of the following values are set:

- TxBD[PAD/CRC] is set in first TxBD
- TxBD[TC] is set in first TxBD

- MACCFG2[PAD/CRC] is set
- MACCFG2[CRC] is set

The Tx_EN is negated after the FCS is sent. This notifies the PHY of the need to generate the illegal Manchester encoding that signifies the end of an Ethernet frame. Following the transmission of the FCS, the Ethernet controller writes the frame status bits into the BD and clears TxBD[R]. If the end of the current buffer is reached and TxBD[L] is cleared (a frame is comprised of multiple buffer descriptors), only TxBD[R] is cleared.

For both half- and full-duplex modes, an interrupt can be issued depending on TxBD[I]. The Ethernet controller then proceeds to the next TxBD in the table. In this way, the core can be interrupted after each frame, after each buffer, or after a specific buffer is sent. If TxBD[PAD/CRC] is set, the Ethernet controller pads any frame shorter than 64 bytes with zero bytes to make up the minimum length.

To pause transmission, or rearrange the transmit queue, set DMACTRL[GTS]. This can be useful for transmitting expedited data ahead of previously-linked buffers or for error situations. If this bit is set, the eTSEC transmitter performs a graceful transmit stop. The Ethernet controller stops immediately if no transmission is in progress or continues transmission until all queued frames in the Tx FIFO have been disposed of. The IEVENT[GTSC] interrupt occurs once the graceful transmit stop operation is completed. After the DMACTRL[GTS] is cleared, the eTSEC resumes transmission with the next frame.

While the eTSEC is in 10/100Mbps mode it sends bytes least-significant nibble first and each nibble is sent lsb first. While it is in 1000Mbps mode it sends bytes LSB first.

### 14.6.3.4 Gigabit Ethernet Frame Reception

The eTSEC Ethernet receiver is designed to work with little core intervention and can perform data extraction, address recognition, CRC checking, short frame checking, and maximum frame-length checking.

After a hardware reset, the software driver clears the RSTAT register and sets MACCFG1[RX_EN]. The Ethernet receiver is enabled and immediately starts processing receive frames. The MAC checks for when TSEC*n*_RX_DV is asserted and as long as TSEC*n*_COL remains negated (full-duplex mode ignores TSEC*n*_COL), the MAC looks for the start of a frame by searching for a valid preamble/SFD (start of frame delimiter) header, which is stripped (unless MACCFG2[PreAM RxEN] is set) and the frame begins to be processed. If a valid header is not found, the frame is ignored.

If the receiver detects the first bytes of a frame, the eTSEC controller begins to perform the frame recognition function through destination address (DA) recognition (see Section 14.6.3.7, “Frame Recognition”). Based on this match the frame can be accepted or rejected. The receiver can filter frames based on individual (unicast), group (multicast), and broadcast addresses. Because Ethernet receive frame data is not written to memory until the internal frame recognition algorithm is complete, system bus usage is not wasted on frames unwanted by this station.

If a frame is accepted, the Ethernet controller fetches the receive buffer descriptor (RxBD) from either queue 0 or the queue determined by the filer. If the RxBD is not being used by software (RxBD[E] is set), the eTSEC starts transferring the incoming frame. RxBD[F] is set for the first RxBD used for any particular receive frame. If the current RxBD is not available for the received frame, a receive busy error condition is raised in IEVENT[BSY].

After the buffer is filled, the eTSEC clears RxBD[E] and, if RxBD[I] is set, generates an interrupt. If the incoming frame is larger than the buffer, the Ethernet controller fetches the next RxBD in the table. If it is empty, the controller continues receiving the rest of the frame. In half-duplex mode, if a collision is detected during the frame, no RxBDs are used; thus, no collision frames are presented to the user except late collisions, which indicate LAN problems.

The RxBD length is determined by the MRBL field in the maximum receive buffer length register (MRBLR). The smallest valid value is 64 bytes, with larger values being be some integral multiple of 64 bytes. During reception, the Ethernet controller checks for frames that are too short or too long. After the frame ends (CRS is negated), the receive CRC field is checked and written to the data buffer. The data length written to the last RxBD in the Ethernet frame is the length of the entire frame, which enables the software to recognize an oversized frame condition.

Receive frames are not truncated when they exceed maximum frame bytes in the MAC’s maximum frame register if MACCFG2[Huge Frame] is set, yet the babbling receiver error interrupt occurs (IEVENT[BABR] is set) and RxBD[LG] is set.

After the receive frame is complete, the Ethernet controller sets RxBD[L], updates the frame status bits in the RxBD, and clears RxBD[E]. If RxBD[I] is set, the Ethernet controller next generates an interrupt (that can be masked) indicating that a frame was received and is in memory. The Ethernet controller then waits for a new frame.

To interrupt reception or rearrange the receive queue, DMACTRL[GRS] must be set. If this bit is set, the eTSEC receiver performs a graceful receive stop. The Ethernet controller stops immediately if no frames are being received or continues receiving until the current frame either finishes or an error condition occurs. The IEVENT[GRSC] interrupt event is signaled after the graceful receive stop operation is completed. While in this mode the user can write to registers that are accessible to both the user and the eTSEC hardware without fear of conflict, and finally clear IEVENT[GRSC]. After DMACTRL[GRS] is cleared, the eTSEC scans the input data stream for the start of a new frame (preamble sequence and start of frame delimiter), it resumes receiving, and the first valid frame received is placed in the next available RxBD.

### 14.6.3.5 Ethernet Preamble Customization

By default eTSEC generates a standard Ethernet preamble sequence prior to transmitting frames. However, the user can substitute a custom preamble sequence for the purpose of controlling switching equipment at the receiver, particularly at 100/1000Mbps speeds. In FIFO mode preamble customization is ignored; in any RMII mode only the standard preamble can be transmitted.

eTSEC normally searches for and discards the standard Ethernet preamble sequence upon receiving frames. Part of the received preamble sequence can be optionally recovered and returned as part of the frame data, making it visible to user software. Note however, that no preamble is received in FIFO mode, and preamble cannot be recovered in any RMII mode. Note that it is also possible for the first two bytes of custom preamble (PreOct0 and PreOct1) to be lost in during conversion to ten-bit code groups in the PCS sub-layer. Thus is it recommended that any custom preamble start at PreOct2.

### 14.6.3.5.1 User-Defined Preamble Transmission

To substitute a custom preamble, the user must ensure that:

- MACCFG2[PreAm TxEN] bit is set
- The first TxBD of every frame containing a custom preamble has its PRE bit set
- An 8-byte custom preamble sequence appears before the Ethernet DA field in the first transmit data buffer

The definition of the 8-byte custom preamble sequence is shown in Figure 14-141.

| Byte Offsets | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0–1 | PreOct0 | | | | | | | | PreOct1 | | | | | | | |
| 2–3 | PreOct2 | | | | | | | | PreOct3 | | | | | | | |
| 4–5 | PreOct4 | | | | | | | | PreOct5 | | | | | | | |
| 6–7 | PreOct6 | | | | | | | | | | | | | | | |

**Figure 14-141. Definition of Custom Preamble Sequence**

The fields of the custom preamble sequence are described in Table 14-150. It should be noted that use of preamble octets matching the standard start of frame delimiter (0xD5) can be expected to trigger premature frame reception by the receiving station.

**Table 14-150. Custom Preamble Field Descriptions**

| Bytes | Bits | Name | Description |
|---|---|---|---|
| 0–1 | 0–7 | PreOct0 | Octet #0 of custom transmit preamble. This is the first octet of preamble sent. |
| | 8–15 | PreOct1 | Octet #1 of custom transmit preamble. This is the second octet of preamble sent. |
| 2–3 | 0–7 | PreOct2 | Octet #2 of custom transmit preamble. This is the third octet of preamble sent. |
| | 8–15 | PreOct3 | Octet #3 of custom transmit preamble. This is the fourth octet of preamble sent. |
| 4–5 | 0–7 | PreOct4 | Octet #4 of custom transmit preamble. This is the fifth octet of preamble sent. |
| | 8–15 | PreOct5 | Octet #5 of custom transmit preamble. This is the sixth octet of preamble sent. |
| 6–7 | 0–7 | PreOct6 | Octet #6 of custom transmit preamble. This is the seventh octet of preamble sent. The last octet (the start of frame delimiter) is generated by the MAC automatically. |
| | 8–15 | — | Reserved; should be cleared. |

### 14.6.3.5.2 User-Visible Preamble Reception

To return the received preamble, the user must ensure that:

- MACCFG2[PreAm RxEN] bit is set
- Space for an 8-byte preamble sequence is allowed before the Ethernet DA field in the first receive data buffer of each frame

The definition of the 8-byte received preamble sequence is shown in Figure 14-142.

| Byte Offsets | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0–1 | PreOct0 | | | | | | | | PreOct1 | | | | | | | |
| 2–3 | PreOct2 | | | | | | | | PreOct3 | | | | | | | |
| 4–5 | PreOct4 | | | | | | | | PreOct5 | | | | | | | |
| 6–7 | PreOct6 | | | | | | | | | | | | | | | |

**Figure 14-142. Definition of Received Preamble Sequence**

The fields of the received preamble sequence are described in Table 14-151. Should the received preamble be shorter than the 7-octet sequence defined by IEEE Std. 802.3, initial bytes of the received preamble sequence hold undefined values. The standard start of frame delimiter (0xD5) is always omitted. Note that preamble extraction is not possible in RMII mode.

**Table 14-151. Received Preamble Field Descriptions**

| Bytes | Bits | Name | Description |
|---|---|---|---|
| 0–1 | 0–7 | PreOct0 | Octet #0 of received preamble. This is the first octet of preamble received. |
| | 8–15 | PreOct1 | Octet #1 of received preamble. This is the second octet of preamble received. |
| 2–3 | 0–7 | PreOct2 | Octet #2 of received preamble. This is the third octet of preamble received. |
| | 8–15 | PreOct3 | Octet #3 of received preamble. This is the fourth octet of preamble received. |
| 4–5 | 0–7 | PreOct4 | Octet #4 of received preamble. This is the fifth octet of preamble received. |
| | 8–15 | PreOct5 | Octet #5 of received preamble. This is the sixth octet of preamble received. |
| 6–7 | 0–7 | PreOct6 | Octet #6 of received preamble. This is the seventh octet of preamble received. The last octet (the start of frame delimiter) is discarded. |
| | 8–15 | — | Reserved |

### 14.6.3.6 RMON Support

Using promiscuous mode, the eTSEC can automatically gather network statistics required for remote network interface monitoring. The RMON MIB group 1, RMON MIB group 2, RMON MIB group 3, RMON MIB group 9, RMON MIB2, and the IEEE 802.3 Ethernet MIB are supported. For RMON statistics and their corresponding counters, see the memory map.

### 14.6.3.7 Frame Recognition

The Ethernet controller performs frame recognition using destination address (DA) recognition. A frame can be rejected or accepted based on the outcome.

#### 14.6.3.7.1 Destination Address Recognition and Frame Filtering

The eTSEC can perform layer 2 frame filtering on the basis of destination Ethernet address (DA), as illustrated by the flowchart in Figure 14-143.

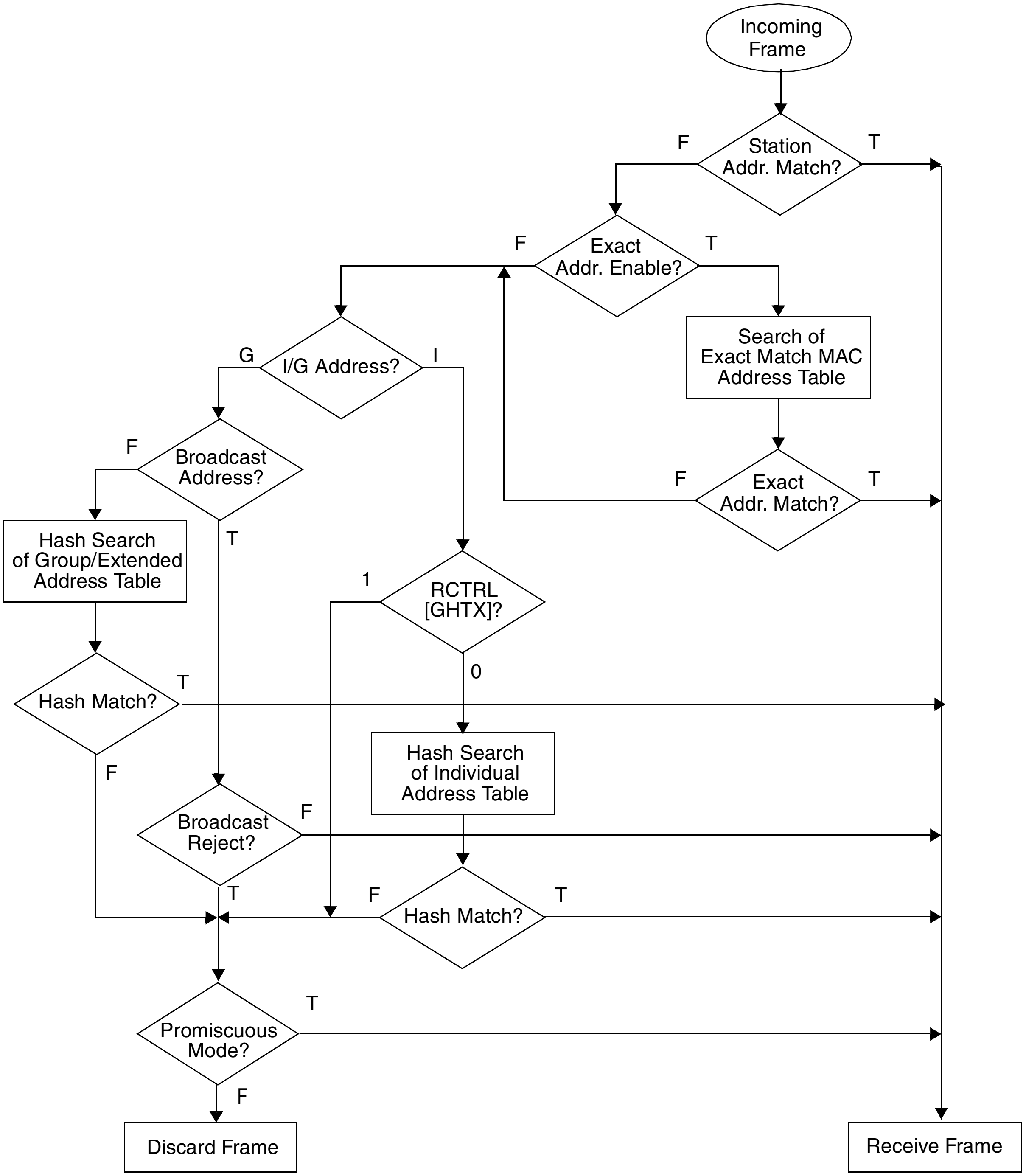


**Figure 14-143. Ethernet Address Recognition Flowchart**

In promiscuous mode, the eTSEC accepts all received frames regardless of DA. Note, however, that Ethernet frame filtering simply restricts the traffic seen by the receive queue filer. Therefore even in promiscuous mode it remains possible to program the filer to reject frames based on their higher-layer header contents.

In the case of an individual address, the DA field of the received frame is compared with the physical address that the user programs in the station address registers (MACSTNADDR1 and MACSTNADDR2). If the DA does not match the station address, and exact MAC address matching is enabled through RCTRL[EMEN], the controller performs address recognition on the multiple MAC addresses written to the MAC*x*ADDR1 and MAC*x*ADDR2 registers. These virtual addresses give a particular eTSEC the ability to mirror other MACs on the network, which caters for router redundancy protocols, such as HSRP and VRRP.

If exact MAC address matching is not enabled, the eTSEC determines whether DA is a group or individual address. If DA is the standard broadcast address, and broadcast addresses are not rejected, the frame is accepted. If any other group address is received, the eTSEC looks-up the DA by means of the group hash table. The group hash table may be extended to 512 entries if RCTRL[GHTX] = 1. Otherwise, an individual address is hashed into the 256-entry individual hash table when RCTRL[GHTX] = 0.

#### 14.6.3.7.2 Hash Table Algorithm

The hash table process used in the group hash filtering operates as follows. By default, the Ethernet controller maps any 48-bit destination address into one of 256 bins, represented by the 256 bits in IGADDR0–IGADDR7 for individual addresses, and the 256 bits in GADDR0–GADDR7 for group addresses. But in the case where RCTRL[GHTX] is set, both sets of registers are combined into an extended group-only hash table of 512 bits, where IGADDR0–IGADDR7 contain the first 256 bits and GADDR0–GADDR7 contain the last 256 bits. No individual-address table exists in extended mode.

The 48-bit destination address received by the MAC is passed through the Ethernet CRC-32 algorithm to produce a hash value. The CRC polynomial used is:

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x + 1$$

The MAC initializes its CRC register to 0xFFFFFFFF before computing a CRC on the 6 bit-reversed octets of the DA. A non-optimized sample of C code for computing the DA hash is listed in Figure 14-144. The 9 most significant bits of the raw, uninverted CRC are used as the hash table index, H[8:0]. If RCTRL[GHTX] = 0, bits H[8:6] select one of the 8 IGADDR or GADDR registers, while bits H[5:1] select a bit within the 32-bit register. If RCTRL[GHTX] = 1, bits H[8:5] select one of the 16 registers in the {IGADDR, GADDR} set, while bits H[4:0] select a bit within the 32-bit register. For example, if H[8:5] = 7, IGADDR7 is selected, whereas H[8:5] = 9 selects GADDR1.

```
/* Wrapper macros for 256-bucket and 512-bucket hash tables:
   Pass 6-byte Ethernet MAC address as parameter. */
#define TSEC_HASH256(macaddr) ((crc32(macaddr) >> 24) & 0xff)
#define TSEC_HASH512(macaddr) ((crc32(macaddr) >> 23) & 0x1ff)

/* CRC constants.  Note: CRC-32 polynomial is bit-reversed. */
#define CRC_POLYNOMIAL 0xedb88320
#define CRC_INITIAL    0xffffffff
#define MAC_ADDRLEN    6
#define BITS_PER_BYTE  8

/* crc32() Takes the array of bytes, macaddr[], representing an
   Ethernet MAC address and returns the CRC-32 result over these bytes,
   where each byte is used in bit-reversed form (Ethernet bit order).
   Index 0 of macaddr[] is the first byte of the address on the wire.
   Test case: the result of crc32 on {0x00, 0x01, 0x02, 0x03, 0x04, 0x05}
   should be 0xad0c28f3.
 */
unsigned long crc32(unsigned char macaddr[MAC_ADDRLEN])
{
  unsigned long crc, result;
  int byte, i;

  /* CRC-32 algorithm starts by inverting first 4 bytes */
  crc = CRC_INITIAL;
  /* add each byte to running CRC accumulator */
  for (byte = 0; byte < MAC_ADDRLEN; ++byte) {
    crc ^= macaddr[byte];
    /* shift CRC right to perform but reversal on byte of address */
    for (i = 0; i < BITS_PER_BYTE; ++i)
      if (crc & 1)
     crc = (crc >> 1) ^ CRC_POLYNOMIAL;
      else
     crc >>= 1;
  }
  /* finally, reverse bits of result to get CRC in normal bit order */
  for (result = 0, i = 4*BITS_PER_BYTE-1; i >= 0; crc >>= 1, --i)
    result |= (crc & 1) << i;
  return result;
}
```

**Figure 14-144. Sample C Code for Computing eTSEC Hash Table Indices**

If the CRC hash table index selects a bit that is set in the hash table, the frame is accepted. If 32 group addresses are stored in the hash table and random group addresses are received, the extended hash table prevents roughly 480/512 (93.8%) of the group address frames from reaching memory. Software must further filter those that reach memory to determine if they contain the correct addresses. Alternatively, small multicast groups can be held in the exact match MAC address registers, which guarantees that only correct frames are admitted.

The effectiveness of the hash table declines as the number of addresses increases. For instance, as the number of addresses stored in the 512-bin hash table increases, the vast majority of the hash table bits are set, preventing only a small fraction of frames from reaching memory.

**NOTE**

The hash table cannot be used to reject frames that match a set of selected addresses because unintended addresses can map to the same bit in the hash table. The receive queue filer may be used to reject frames with unintended address hits in the hash table.

### 14.6.3.8 Magic Packet Mode

eTSEC implements the AMD Magic Packet™ specification for LAN-initiated power management. This mode is normally entered with the rest of the system in a low-power sleep mode. Software must enable normal receive function in the Ethernet MAC, and then finally set the MACCFG2[MPEN] bit to enable Magic Packet detection before the system enters a reduced mode. While the rest of the system is operating in low-power mode, the enabled eTSEC continues to receive Ethernet frames, but discards them immediately. Upon receipt of any frame whose contents contain the valid Magic Packet sequence, the eTSEC exits out of Magic Packet mode, thus clearing MACCFG2[MPEN], and raises an error/diagnostic interrupt through IEVENT[MAG], which causes the surrounding system to wake-up. Frames received after Magic Packet mode has exited are received into software buffers as usual. Software can abort Magic Packet mode by writing 0 to MACCFG2[MPEN] at any time.

AMD specify a Magic Packet™ to be any Ethernet frame containing a valid Ethernet header (Destination and Source Addresses) and valid FCS (CRC-32), and whose payload includes the specific Magic Packet byte sequence at any offset from the start of frame. The specific byte sequence comprises an unbroken stream of 102 bytes, the first 6 bytes of which are 0xFFFFFF_FFFFFF, followed by 16 copies of the MAC’s unique IEEE station address in the normal byte order for Ethernet addresses. For example, if the station address were 0x112233_445566, then the MAC would have to receive 0xFFFFFF_FFFFFF, 0x112233_445566, ..., 0x112233_445566 in any payload to detect a Magic Packet. Only frames addressed specifically to the MAC’s station address or a valid multicast or broadcast address can be examined for the Magic Packet sequence.

### 14.6.3.9 Flow Control

Because collisions cannot occur in full-duplex mode, gigabit Ethernet can operate at the maximum rate. If the rate becomes too fast for a station’s receiver, the station’s transmitter can send flow-control frames to reduce the rate. Flow-control instructions are transferred by special frames of minimum frame size. The length/type fields of these frames have a special value.

Table 14-152 lists the flow-control frame structure.

**Table 14-152. Flow Control Frame Structure**

| Size [Octets] | Description | Value | Comment |
|---|---|---|---|
| 7 | Preamble | | — |
| 1 | SFD | | Start frame delimiter |
| 6 | Destination address | 01-80-C2-00-00-01 | Multicast address reserved for use in MAC frames (or MAC station address) |
| 6 | Source address | | — |
| 2 | Length/type | 88-08 | Control frame type |

Table 14-152. Flow Control Frame Structure (continued)

| Size [Octets] | Description | Value | Comment |
|---|---|---|---|
| 2 | MAC opcode | 00-01 | Pause command |
| 2 | MAC parameter | | Pause time as defined by the PTV[PT] field. The pause period is measured in pause_quanta, a speed independent constant of 512 bit-times (unlike slot time). The most-significant octet is transmitted first. |
| 2 | Extended MAC parameter | | Pause time extended as defined by the PTV[PTE] field. The most significant octet is transmitted first. |
| 40 | Reserved | — | — |
| 4 | FCS | | Frame check sequence (CRC) |

If flow-control mode is enabled (MACCFG1[Rx_Flow] is set) and the receiver identifies a pause-flow control frame, transmission stops for the time specified in the control frame. The controller completes any frame in progress before stopping transmission and does not commence counting the pause time until transmit is idle. During a pause, only a control frame can be sent (TCTRL[TFC_PAUSE] is set). Normal transmission resumes after the pause timer stops counting, or resumes immediately if a pause frame with a zero time-out is received. If another pause-control frame is received during the pause, the period changes to the new value received.

### 14.6.3.10 Interrupt Handling

The following describes what usually occurs within a eTSEC interrupt handler:

- If an interrupt occurs, read IEVENT to determine interrupt sources. IEVENT bits to be handled in this interrupt handler are normally cleared at this time. There are three kinds of interrupts:
  - — Receive data frame interrupts, when bits RXB or RXF in IEVENT are set
  - — Transmit data frame interrupts, when bits TXB or TXF in IEVENT are set
  - — Error, diagnostic, and special interrupts (all bits in IEVENT other than RXB, RXF, TXB, or TXF)
- Process the TxBDs to reuse them if the IEVENT[TXB, TXF or TXE] were set. Consult register bits TSTAT[TXF0–TXF7] to determine which TxBD rings gave rise to the transmit interrupt in the case of TXF. If the transmit speed is fast or the interrupt delay is long, more than one transmit buffer may have been sent by the eTSEC; thus, it is important to check more than just one TxBD during the interrupt handler. One common practice is to process all TxBDs in the interrupt handler until one is found with R set.
- Obtain data from RxBD rings if IEVENT[RXC, RXB or RXF] is set. Consult register bits RSTAT[RXF0–RXF7] to determine which RxBD rings gave rise to the receive interrupt in the case of RXF. If the receive speed is fast or the interrupt delay is long, the eTSEC may have received more than one RxBD; thus, it is important to check more than just one RxBD during interrupt handling. Typically, all RxBDs in the interrupt handler are processed until one is found with E set. Because the eTSEC pre-fetches BDs, the BD table must be big enough so that there is always another empty BD to pre-fetch, otherwise a BSY error occurs.

- Clear any set halt or frame interrupt bits in TSTAT and RSTAT registers, or DMACTRL[GTS] and DMACTRL[GRS] by writing 1s to these bits.
- Continue normal execution.

**Table 14-153. Non-Error Transmit Interrupts**

| Interrupt | Description | Action Taken by the eTSEC |
|---|---|---|
| GTSC | Graceful transmit stop complete: transmitter is put into a pause state after completion of the frame currently being transmitted. | None |
| TXC | Transmit control: Instead of the next transmit frame, a control frame was sent. | None |
| TXB | Transmit buffer: A transmit buffer descriptor, that is not the last one in the frame, was updated in one of the enabled TxBD rings. | Programmable 'write with response' TxBD to memory before setting IEVENT[TXB]. |
| TXF | Transmit frame: A frame from an enabled TxBD ring was transmitted and the last transmit buffer descriptor (TxBD) of that frame was updated. | Programmable 'write with response' to memory on the last TxBD before setting IEVENT[TXF]. |

**Table 14-154. Non-Error Receive Interrupts**

| Interrupt | Description | Action Taken by the eTSEC |
|---|---|---|
| GRSC | Graceful receive stop complete: Receiver is put into a pause state after completion of the frame currently being received. | None |
| RXC | Receive control: A control frame was received. As soon as the transmitter finishes sending the current frame, a pause operation is performed. | None |
| RXB | Receive buffer: A receive buffer descriptor, that is not the last one of the frame, was updated in one of the enabled RxBD rings. | Programmable 'write with response' RxBD to memory before setting IEVENT[RXB]. |
| RXF | Receive frame: A frame was received to an enabled RxBD ring and the last receive buffer descriptor (RxBD) of that frame was updated. | Programmable 'write with response' to memory on the last RxBD before setting IEVENT[RXF]. |

#### 14.6.3.10.1 Interrupt Coalescing

Interrupt coalescing offers the user the ability to contour the behavior of the eTSEC with regard to frame interrupts. Separate but identical mechanisms exist for both transmitted frames and received frames. In either case, frame interrupts require that software set the I-bit in RxBDs or TxBDs, and disable buffer interrupts (IEVENT[RXB] or IEVENT[TXB]). Particular rings can remain free of interrupts by ensuring that the I-bit is consistently cleared in all BDs. While interrupt coalescing is enabled, a transmit or receive frame interrupt is raised either when a counter threshold-defined number of frames is received/transmitted or the timer threshold-defined period of time has elapsed, whichever occurs first. Disabling and then re-enabling interrupt coalescing forces reset of the coalescing timers and counters to reflect changes made to the threshold registers.

#### 14.6.3.10.2 Interrupt Coalescing By Frame Count Threshold

To avoid interrupt bandwidth congestion due to frequent, consecutive interrupts, the user may enable and configure interrupt coalescing to deliberately group frame interrupts, reducing the total number of interrupts raised. The number of frames received or transmitted prior to an interrupt being raised is determined by the frame threshold field (ICFT) in the appropriate interrupt coalescing configuration register (RXIC or TXIC). The frame threshold field may be assigned a value between 1 and 255. The internal transmit or receive frame counter decrements from this initial value each time a frame is transmitted or received. Upon reaching zero, an interrupt is raised, the appropriate threshold counter is reset to the value in the ICFT field, and then eTSEC continues counting frames while the interrupt is active. The appropriate threshold counter is also reset to the value in the ICFT field if an interrupt is raised subject to the corresponding threshold timer.

#### 14.6.3.10.3 Interrupt Coalescing By Timer Threshold

To avoid stale frame interrupts, the user may also assign a timer threshold, beyond which any frame interrupts not yet raised are forced. The timer threshold fields of the receive and transmit interrupt coalescing configuration registers (RXIC[ICTT] and TXIC[ICTT]) are defined in units equivalent to 64 interface clocks or system clocks, depending on the setting of the ICCS field in RXIC and TXIC.

After transmitting a frame, the transmit interrupt coalescing threshold time begins counting down from the value in TXIC[ICTT]. An interrupt is raised when the counter reaches zero. In the event of graceful transmit stop completion before the coalescing timer expires, the eTSEC issues two interrupts, the first for GTS, the second for TXF (due to timer expiration of a pending event). To prevent the second interrupt from affecting servicing of the GTS event, it is recommended that the user mask out the TXF event during execution of the service routine. After receiving a frame, the receive interrupt coalescing threshold time begins counting down from the value in RXIC[ICTT]. An interrupt is raised when the counter reaches zero. In the event of graceful receive stop completion before the coalescing timer expires, the eTSEC issues two interrupts, the first for GRS, the second for RXF (due to timer expiration of a pending event). To prevent the second interrupt from affecting servicing of the GRS event, it is recommended that the user mask out the RXF event during execution of the service routine.

The interrupt coalescing timer thresholds (transmit and receive, operating independently) may be values ranging from 0x0001 to 0xFFFF. Table 14-155 specifies the range of possible timing thresholds subject to timer clock source, the interface or system frequency, and the value of the RXIC[ICTT] or TXIC[ICTT] field.

**Table 14-155. Interrupt Coalescing Timing Threshold Ranges**

| ICCS (Clock Source) | eTSEC Interface Format and Frequency or eTSEC System Frequency | Interrupt Coalescing Threshold Time | |
|---|---|---|---|
| | | Minimum (ICTT = 0x0001) | Maximum (ICTT = 0xFFFF) |
| 0 (I/F clock) | 10Base-T at 2.5 MHz | 25.6 µs | 1.68 s |
| 0 (I/F clock) | 100Base-T at 25 MHz | 2.56 µs | 168 ms |
| 0 (I/F clock) | 1000Base-T at 125 MHz | 0.51 µs | 33.6 ms |
| 1 (sys. clock) | eTSEC operating at 266 MHz | 0.24 µs | 15.7 ms |
| 1 (sys. clock) | eTSEC operating at 333 MHz | 0.19 µs | 12.6 ms |

The transmit timer threshold counter is reset to the value in TXIC[ICTT] and begins counting down on transmission of the frame following an interrupt.

The receive timer threshold counter is reset to the value in RXIC[ICTT] and begins counting down on receiving the frame following an interrupt.

### 14.6.3.11 Inter-Frame Gap Time

If a station must transmit, it waits until the LAN becomes silent for a specified period (inter-frame gap, or IFG). The minimum inter-packet gap (IPG) time for back-to-back transmission is set by IPGIFG[Back-to-Back Inter-Packet-Gap]. The receiver receives back-to-back frames with the minimum interframe gap (IFG) as set in IPGIFG[Minimum IFG Enforcement]. If multiple frames are ready to transmit, the Ethernet controller follows the minimum IPG as long as the following restrictions are met:

- The next transmit buffer descriptor address (TBPTRn) for a ring is located at a 16-byte aligned address when the ring starts transmitting.
- All BDs for any multiple-BD frame reside in the same cache line.
- TCP/UDP and IP Checksum generation are disabled in each frame's TxFCB, or in TCTRL, or frames are limited to 1200 bytes in length.
- Each TxBD[Data Length] >= 64 bytes.

If the TxBD alignment restrictions are not met, the back-to-back IPG may be as many as 32 cycles due to BD refetching. If the TxBD size restriction is not met, the back-to-back IPG may be significantly longer.

In half-duplex mode, after a station begins sending, it continually checks for collisions on the LAN. If a collision is detected, the station forces a jam signal (all ones) on its frame and stops transmitting. Collisions usually occur close to the beginning of a packet. The station then waits a random time period (back-off) before attempting to send again. After the back-off completes, the station waits for silence on the LAN (carrier sense negated) and then begins retransmission (retry) on the LAN. Retransmission begins 36 bit times after carrier sense is negated for at least 60 bit times. If the frame is not successfully sent within a specified number of retries, an error is indicated (collision retry limit exceeded).

If a queue is actively transmitting, and multiple BDs are ready to transmit, the ethernet controller satisfies the minimum IPG (96 bit times) as long as the following restrictions are met by software:

- The next BD is always ready when fetched by the controller.
- Frames use a single BD.
- TCP/UDP and IP Checksum generation are disabled in each frame's TxFCB, or in TCTRL, or frames are limited to 1200 bytes in length.
- Each TxBD[Data Length] >= 64 bytes.

If multiple BDs per frame are used, BD fetching may result in a gap between frames of up to 32 cycles if the fetch delay causes the amount of data in the TxFIFO to fall below the transmit threshold (1 KB).

If TCP Offload is enabled (TCTRL[TUCSEN]=1 or TCTRL[IPCSEN]=1), the entire frame must be loaded in the TxFIFO before transmission can start. For frames longer than 1200 bytes, this delay in start of transmission can result in extra inter-packet gaps, with the delay increasing with size of frame.

#### 14.6.3.12 Internal and External Loop Back

Setting MACCFG1[Loop Back] causes the MAC transmit outputs to be looped back to the MAC receive inputs. Clearing this bit results in normal operation. This bit is cleared by default. Clearing this bit results in normal operation.

#### 14.6.3.13 Error-Handling Procedure

The eTSEC reports frame reception and transmission error conditions using the channel BDs, the error counters, and the IEVENT register.

Transmission errors are described in Table 14-156.

**Table 14-156. Transmission Errors**

| Error | Response |
|---|---|
| Transmitter underrun | Transmitter underrun can occur either after frame transmission has commenced, or in response to an incomplete sequence of TxBDs. In the former case, the controller sends 32 bits that ensure a CRC error, and terminates buffer transmission. In the latter case, the relevant transmit queue is halted. In all cases, the eTSEC closes the buffer, sets TxBD[UN], IEVENT[XFUN], and IEVENT[TXE]. The controller resumes transmission after TSTAT[THLT] is cleared (and DMACTRL[GTS] is cleared). |
| Retransmission attempts limit expired | The controller terminates buffer transmission, sets TxBD[RL], closes the buffer, IEVENT[CRL], and IEVENT[TXE]. Transmission resumes after TSTAT[THLT] is cleared (and DMACTRL[GTS] is cleared). |
| Late collision | The controller terminates buffer transmission, sets TxBD[LC], closes the buffer, IEVENT[LC], and IEVENT[TXE]. The controller resumes transmission after TSTAT[THLT] is cleared (and DMACTRL[GTS] is cleared). |
| Memory read error | A system bus error occurred during a DMA transaction. The controller sets IEVENT[EBERR], DMA stops sending data to the FIFO which causes an underrun error, and therefore TxBD[UN] is set, but IEVENT[XFUN] is not set. The TSTAT[THLT] is set. Transmits are continued once TSTAT[THLT] is cleared. |
| Data parity error | Data in the transmit FIFO was potentially corrupted. The controller sets IEVENT[DPE], but otherwise continues transmission until halted explicitly. |
| Babbling transmit error | A frame is transmitted which exceeds the MAC's Maximum Frame Length and MACCFG2[Huge Frame] is a 0. The controller sets IEVENT[BABT] and continues without interruption. |

Reception errors are described in Table 14-157.

**Table 14-157. Reception Errors**

| Error | Description |
|---|---|
| Overrun error | The Ethernet controller maintains an internal FIFO buffer for receiving data. If a receiver FIFO buffer overrun occurs, the controller sets RxBD[OV], sets RxBD[L], closes the buffer, increments the discarded frame counter (RDRP), and sets IEVENT[RXF], The receiver then enters hunt mode (seeking start of a new frame). |
| Busy error | A frame is received and discarded due to a lack of buffers. The controller sets IEVENT[BSY] and increments the discarded frame counter (RDRP). In addition, the RSTAT[QHLT*n*] bit is set. RDRP increments for each frame that is received while the receiver is halted due to a busy condition. The halted queue resumes reception once the RSTAT[QHLT*n*] bit is cleared. |

**Table 14-157. Reception Errors (continued)**

| Error | Description |
|---|---|
| Filed frame to invalid queue error | A frame is received and discarded as a result of the filer directing it to an RxBD ring that is currently not enabled. The controller sets IEVENT[FIQ] and increments the discarded frame counter (RDRP). |
| Parser error | If the receive frame parser is enabled, a parse error can be flagged as a result of inconsistencies discovered between fields of the embedded packet headers. For example, the L2 header may indicate an IPv4 header, but the IP version number fails to match. In the event of a parse error, parsing is terminated at the inconsistent header, and the RxFCB[PERR] field indicates at which layer of the protocol stack the error was discovered. Receiver function continues regardless of parse errors, but IEVENT[PERR] is set. The receive queue filer may operate with reduced or default information in some cases; therefore, filer rule sets should be constructed so as to be tolerant of misformed frames.<br>**Note:** Any values in the length/type field between 1500 and 1536 is treated as a length, however, only illegal packets exist with this length/type since these are not valid lengths and not valid types. These are treated by the MAC logic as out of range.<br>Software must confirm the parser and filer results by checking the type/length field after the packet has been written to memory to see if it falls in this range. |
| Non-octet error (dribbling bits) | The Ethernet controller handles a nibble of dribbling bits if the receive frame terminates as non-octet aligned and it checks the CRC of the frame on the last octet boundary. If there is a CRC error, the frame non-octet aligned (RxBD[NO]) error is reported, IEVENT[RXF] is set, and the alignment error counter increments. The eTSEC relies on the statistics collector block to increment the receive alignment error counter (RALN). If there is no CRC error, no error is reported. |
| CRC error | If a CRC error occurs, the controller sets RxBD[CR], closes the buffer, and sets IEVENT[RXF]. This eTSEC relies on the statistics collector block to record the event. After receiving a frame with a CRC error, the receiver then enters hunt mode. |
| Memory read error | A system bus error occurred during a DMA transaction. The controller sets IEVENT[EBERR] and discards the frame and increments the discarded frame counter (RDRP). In addition the RSTAT[QHLT*n*] bit is set. The halted queue resumes reception once the RSTAT[QHLT*n*] bit is cleared. |
| Data parity error | Data in the receive FIFO or filer table was potentially corrupted. The controller sets IEVENT[DPE], but otherwise continues reception until halted explicitly. |
| Babbling receive error | A frame is received that exceeds the MAC's maximum frame length. The controller sets IEVENT[BABR] and continues. |

## 14.6.4 TCP/IP Off-Load

Each eTSEC provides hardware support for accelerating the basic functions of TCP/IP packet transmission and reception. By default, these features are disabled and must be explicitly enabled through RCTRL and TCTRL. In this configuration, the eTSEC processes frames as vanilla Ethernet frames and none of the multi-ring QoS/CoS receive services or per-frame VLAN insertion and deletion are available. Operate eTSEC in this default configuration when using existing TCP/IP stack software that has not been modified to take advantage of TOE.

TOE can be enabled independently for Rx and Tx and at various levels. Receive TOE functions are controlled by RCTRL and transmit functions through a combination of TCTRL[TUCSEN] and the Tx frame control block.

On receive, according to RCTRL[PRSDEP], eTSEC can parse frames at layer 2 of the stack only (Ethernet headers and switching headers), layers 2 to 3 (including IPv4 or IPv6), or layers 2 to 4 (including TCP and

UDP). TOE provides protocol header recognition, header verification (IPv4 header checksum verification), and TCP/UDP payload checksum verification including verification of associated pseudo-header checksums. For large frames off-load of checksum verification saves a significant fraction of the CPU cycles that would otherwise be spent by the TCP/IP stack. IP packet fragmentation and re-assembly, and TCP stream establishment and tear-down are not performed in hardware. The frame parser sets RQFPR[IPF] status flag encountering a fragmented frame. The frame parser in eTSEC searches a maximum of 512 bytes from the start of a received frame when attempting to locate headers; headers deeper than 512 bytes are assumed not to exist, and any associated receive status flags in the frame control block remain cleared.

On transmit, TOE provides IPv4 and TCP/UDP header checksum generation. Like receive TOE, checksum generation reduces CPU load significantly for TCP/IP stacks modified to exploit eTSEC TOE functions. The eTSEC does not checksum transmitted packets with IPv6 routing headers or calculate TCP/UDP checksums from IP fragments. If a transmitted TCP segment requires checksum generation but IPv6 extension headers would prevent eTSEC from calculating the pseudo-header checksum, software can calculate just the pseudo-header checksum in advance and supply it to the eTSEC as part of per-frame TOE configuration.

### 14.6.4.1 Frame Control Blocks

Frame control blocks (FCBs) are 8-byte blocks of TOE control and/or status data that are passed between software (driver and TCP/IP stack) and each eTSEC. A FCB always precedes the frame it applies to, and is present only when TOE functions are being used. As Figure 14-145 shows, the first BD of each frame points to the initial data buffer and the FCB. The initial data buffer must be at least 8 bytes long to contain the FCB without breaking it. Custom or received Ethernet preamble sequences also follow the FCB if preambles are visible.

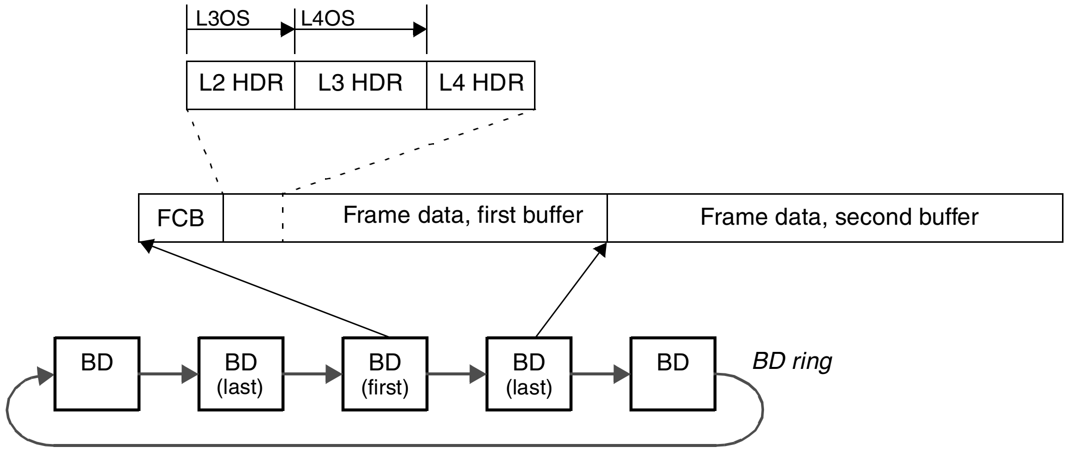


**Figure 14-145. Location of Frame Control Blocks for TOE Parameters**

For TxBD rings, FCBs are assumed present when the TxBD[TOE/UN] bit is set by user software. The eTSEC ignores the TxBD[TOE/UN] bit in all BDs other than those pointing to initial data buffers, therefore FCBs must not be inserted in second and subsequent data buffers. Since TxBD[TOE/UN] can be set under software discretion, TOE acceleration for transmit may be applied on a frame-by-frame basis.

In the case of RxBD rings, FCBs are inserted by the eTSEC whenever RCTRL[PRSDEP] is set to a non-zero value. Only one FCB is inserted per frame, in the buffer pointed to by the RxBD with bit F set. TOE acceleration for receive is enabled for all frames in this case.

### 14.6.4.2 Transmit Path Off-Load and Tx PTP Packet Parsing

TOE functions for transmit are defined by the contents of the Tx FCB. Figure 14-146 describes the definition for the Tx FCB.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Offset + 0 | VLN | IP | IP6 | TUP | UDP | CIP | CTU | NPH | | | | | | | | PTP |
| Offset + 2 | L4OS | | | | | | | | L3OS | | | | | | | |
| Offset + 4 | PHCS | | | | | | | | | | | | | | | |
| Offset + 6 | VLCTL | | | | | | | | | | | | | | | |

**Figure 14-146. Transmit Frame Control Block**

The user instructs the Tx packet to be timestamped via setting bit 15 in the TxFCB to mark a PTP packet. TxFCB[VLCTL] can be translated as the Tx PTP packet identification number. BD[TOE] has to be set to enable transmit PTP packet time stamping. TxFCB[PTP] bit takes precedence over TxFCB[VLN] bit. It disables per packet VLAN tag insertion. On a PTP packet, VLAN tag can be inserted from the DFVLAN register. A proposed TxFCB update for the PTP packet is shown in Figure 14-153.

The contents of the Tx FCB are defined in Table 14-158.

**Table 14-158. Tx Frame Control Block Description**

| Bytes | Bits | Name | Description |
|---|---|---|---|
| 0–1 | 0 | VLN | VLAN control word valid. This bit is ignored when the PTP bit is set. VLAN tag is read from the DFVLAN register if PTP=1.<br>0 Ignore VLCTL field.<br>1 If VLAN tag insertion is enabled for eTSEC, use the VLCTL field as the VLAN control word. |
| | 1 | IP | Layer 3 header is an IP header.<br>0 Ignore layer 3 and higher headers.<br>1 Assume that the layer 3 header is an IPv4 or IPv6 header, and take L3OS field as valid. |
| | 2 | IP6 | IP header is IP version 6. Valid only if IP = 1.<br>0 IP header version is 4.<br>1 IP header version is 6. |
| | 3 | TUP | Layer 4 header is a TCP or UDP header.<br>0 Do not process any layer 4 header.<br>1 Assume that the layer 4 header is either TCP or UDP (see UDP bit), and offload checksumming on the basis that the IP header has no extension headers. |
| | 4 | UDP | UDP protocol at layer 4.<br>0 Layer 4 protocol is either TCP (if TUP = 1) or undefined.<br>1 Layer 4 protocol is UDP if TUP = 1. |

**Table 14-158. Tx Frame Control Block Description (continued)**

| Bytes | Bits | Name | Description |
|---|---|---|---|
| 0–1 | 5 | CIP | Checksum IP header enable.<br>0 Do not generate an IP header checksum.<br>1 Generate an IPv4 header checksum. |
| | 6 | CTU | Checksum TCP or UDP header enable.<br>0 Do not generate a TCP or UDP header checksum. RFC 768 advises that UDP packets not requiring checksum validation should have their checksum field set to zero.<br>1 Generate a TCP header checksum if IP = 1 and TUP = 1 and UDP = 0. |
| | 7 | NPH | Disable calculation of TCP or UDP pseudo-header checksum. This bit should be set if IP options need to be consulted in forming the pseudo-header checksum, as eTSEC does not examine IP options or extension headers for TCP/IP offload on transmit.<br>0 Calculate TCP or UDP pseudo-header checksum as normal, assuming that the IP header has no options.<br>1 Do not calculate a TCP or UDP pseudo-header checksum, but instead use the value in field PHCS when determining the overall TCP or UDP checksum. |
| | 8–14 | — | Reserved |
| | 15 | PTP | Indication to the transmitter that this is a PTP packet. Enabling PTP disables per packet VLAN tag insertion. Instead, VLAN tag will be read from the DFVLAN when the PTP field is true.<br>0 Do not attempt to capture transmission event time<br>1 Valid PTP_ID field. When this packet is transmitted, capture the time of transmission. Must be clear if TMR_CTRL[TE] is clear. |
| 2–3 | 0–7 | L4OS | Layer 4 header offset from start of layer 3 header. The layer 4 header starts L4OS octets after the layer 3 header if it is present. The maximum layer 3 header length supported is thus 255 bytes, which may prevent TCP/IP offload on particularly large IPv6 headers. |
| | 8–15 | L3OS | Layer 3 header offset from start of frame not including the 8 bytes for this FCB. The layer 3 header starts L3OS octets from the start of the frame including any custom preamble header that may be present. The maximum layer 2 header length supported is thus 255 bytes. |
| 4–5 | 0–15 | PHCS | Pseudo-header checksum (16-bit one's complement sum with carry wraparound, but without result inversion) for TCP or UDP packets, calculated by software. Valid only if NPH = 1. |
| 6–7 | 0–15 | VLCTL/<br>PTP_ID | VLAN control word for insertion in the transmitted VLAN tag. Valid only if VLN = 1.Tx PTP packet identification number. This number will be copied into the Tx PTP packet time stamp identification field. PTP field takes precedence over VLN field. |

## 14.6.4.3 Receive Path Off-Load

Upon receive, the Rx FCB returns the status of frame parse and TOE functions applied to the accompanying frame. Figure 14-147 describes the definition for the Rx FCB.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Offset + 0 | VLN | IP | IP6 | TUP | CIP | CTU | EIP | ETU | — | | | | PERR | | — | GPFP |
| Offset + 2 | | | RQ | | | | | | PRO | | | | | | | |
| Offset + 4 | | | | | | | | | | | | | | | | |
| Offset + 6 | VLCTL | | | | | | | | | | | | | | | |

**Figure 14-147. Receive Frame Control Block**

The contents of the Rx FCB are defined in Table 14-159.

Table 14-159. Rx Frame Control Block Descriptions

| Bytes | Bits | Name | Description |
|---|---|---|---|
| 0–1 | 0 | VLN | VLAN tag recognized. This bit is set only if RCTRL[VLEX] is set.<br>0 No VLAN tag recognized.<br>1 IEEE Std. 802.1Q VLAN tag found; VLAN control word in VLCTL is valid. |
| | 1 | IP | IP header found at layer 3.RCTRL[PRSDEP] must be set to 10 or 11 in order to enable IP discovery. See also IP6 bit of FCB.<br>0 No layer 3 header recognized.<br>1 An IP header was recognized at layer 3; the IANA protocol identifier for the next header can be found in PRO; see PRO for more information.<br>If S/W is relying on the RxFCB for the parse results, any RxFCB[IP] bits set with the corresponding RxFCB[PRO] = 0xFF indicates a fragmented packet (or that this packet had a back-to-back IPv6 routing extension header). Additionally, RQFPR[IPF] (see Section 14.5.3.3.8, “Receive Queue Filer Table Property Register (RQFPR)”) indicates that the packet was fragmented. |
| | 2 | IP6 | IP version 6 header found at layer 3.<br>0 No IPv6 header was found.<br>1 The layer 3 header was an IPv6 header provided IP = 1. |
| | 3 | TUP | TCP or UDP header found at layer 4. RCTRL[PRSDEP] must be set to 10 or 11 in order to enable TCP/UDP discovery.<br>0 No layer 4 header recognized.<br>1 The layer 4 header was recognized as either TCP (PRO = 0x06) or UDP (PRO = 0x11). |
| | 4 | CIP | IPv4 header checksum checked. RCTRL[PRSDEP] must be set to 10 or 11 in order to enable IPv4 checksum verification.<br>0 IPv4 header checksum not verified, either because verification was disabled or a valid IPv4 header could not be located.<br>1 IPv4 header checksum was verified by the eTSEC, and bit EIP indicates result. |
| | 5 | CTU | TCP or UDP header checksum checked. RCTRL[PRSDEP] must be set to 11 in order to enable layer 4 checksum verification.<br>0 TCP or UDP header checksum not verified, either because verification was disabled or a valid TCP or UDP header could not be located. If a UDP header with zero checksum was located, this bit is cleared in accordance with RFC 768.<br>1 TCP or UDP header checksum was verified by the eTSEC, and ETU indicates result. |
| | 6 | EIP | IPv4 header checksum verification error. Not valid unless CIP = 1.<br>0 No checksum error in IPv4 header.<br>1 Error in header checksum only if IP = 1 and IP6 = 0. |
| 0–1 | 7 | ETU | TCP or UDP header checksum verification error. Not valid unless CTU = 1.<br>0 No checksum error in TCP or UDP header.<br>1 Error in header checksum only if PRO = 0x06 or PRO = 0x11. |
| | 8–11 | — | Reserved |
| | 12–13 | PER | Parse error.<br>00 No error in L2 to L4 parse<br>01 Reserved<br>10 Inconsistent or unsupported L3 header sequence<br>11 Reserved |
| | 14 | — | Reserved |
| | 15 | GPFP | General-purpose filer event packet. This packet was filed based on matching a GPI rule sequence. |

Table 14-159. Rx Frame Control Block Descriptions (continued)

| Bytes | Bits | Name | Description |
|---|---|---|---|
| 2–3 | 0–1 | — | Reserved |
| | 2–7 | RQ | Receive queue index. This index was selected by the eTSEC Rx Filer (from a matching Filer rule's RQCTRL[Q] field) when it accepted the associated frame. If filing is not enabled, RQ is zero. Note that the 3 least significant bits of RQ correspond with the RxBD ring index whenever RCTRL[FSQEN] = 0. |
| | 8–15 | PRO | If IP = 1, PRO is set as follows:<br>• PRO=0xFF for a fragment header or a back to back route header<br>• PRO=0x*nn* for an unrecognized header, where *nn* is the next protocol field<br>• PRO=(TCP/UDP header), as defined in the IANA specification, if TCP or UDP header is found<br>If IP = 0, PRO is undefined.<br>Note that the eTSEC parser logic stops further parsing when encountering an IP datagram that has indicated that it has fragmented the upper layer protocol. This in general means that there is likely no layer 4 header following the IP header and extension headers. eTSEC leaves the RxFCB[PRO] and RQFPR[L4P] fields 0xFF in this case, which usually means that there was no IP header seen. In this case RxFCB[IP] and optionally RxFCB[IP6] is set. IP header checksumming operates and performs as intended. Most of the time, the eTSEC updates the RxFCB[PRO] field and RQFPR[L4P] fileds with whatever value was found in the protocol field of the IP header. See Section 14.5.3.3.8, "Receive Queue Filer Table Property Register (RQFPR)," for a description of RQFPR. |
| 4–5 | 0–15 | — | Reserved |
| 6–7 | 0–15 | VLCTL | VLAN control word as per IEEE Std. 802.1Q. The lower 12 bits comprise the VLAN identifier. Valid only if VLN = 1. |

## 14.6.5 Quality of Service (QoS) Provision

This section describes the quality of service support features of this device. It includes a parser which extracts vital packet properties and passes then to the filer which essentially acts as a frame classifier.

### 14.6.5.1 Receive Parser

The receive parser parses the incoming frame data and generates filer properties and frame control block (FCB). The receive parser composes of the Ethernet header parser and L3/L4 parser.

The Ethernet header parser parses only L2 (ethertype) headers. It is enabled by RCTRL[PRSDEP] != 0. It has the following key features:

- Extraction of 48-bit MAC destination and source addresses
- Extraction and recognition of the first 2-byte ethertype field
- Extraction and recognition of the final 2-byte ethertype field
- Extraction of 2-byte VLAN control field
- Walk through MPLS stack and find layer 3 protocol
- Walk through VLAN stack and find layer 3 protocol
- Recognition of the following ethertypes for inner layer parsing
  - — LLC and SNAP header

- — JUMBO and SNAP header
- — IPV4
- — IPV6
- — VLAN
- — MPLSU/MPLSM
- — PPPOES
- — ARP

For stack L2 (that is, more than one ethertypes) header, the Ethernet parser traverses through the header until it finds the last valid ethertype or the ethertype is unsupported. Table 14-160 describes what the Ethernet header parser recognizes for stack L2 header.

**Table 14-160. Supported Stack L2 Ethernet Headers**

| Column—Current L2 Ethertype<br>Row—Next Supported L2 Ethertype | LLC/ SNAP | JUMBO/ SNAP | IPV4 | IPV6 | VLAN | MPLSU | MPLSM | PPOES | ARP |
|---|---|---|---|---|---|---|---|---|---|
| **LLC/SNAP** | N | N | Y | Y | Y | Y | Y | Y | Y |
| **JUMBO/SNAP** | N | N | Y | Y | Y | Y | Y | Y | Y |
| **IPV4** | N | N | N | N | N | N | N | N | N |
| **IPV6** | N | N | N | N | N | N | N | N | N |
| **VLAN** | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| **MPLSU** | N | N | Y* | Y* | N | y | Y | N | N |
| **MPLSM** | N | N | Y* | Y* | N | Y | Y | N | N |
| **PPOES** | N | N | Y | Y | N | Y | Y | N | N |
| **ARP** | N | N | N | N | N | N | N | N | N |
| **Note:** * means that it is the next protocol | | | | | | | | | |

The L3 parser is enabled by RCTRL[PRSDEP] = 10 or 11. It begins when the Ethernet parser ends and a valid IPv4/v6 ethertype is found. The L4 header is enabled by RCTRL[PRSDEP] = 11. It begins when the L3 parser ends and a valid TCP/UDP next protocol is found and no fragment frame is found. The primary functionalities of L3(IPv4/6) and L4(TCP/UDP) parsers are as follows:

- IP recognition (v4/v6, ARP, encapsulated protocol)
- IP header checksum verification
- IPv4/6 over IPv4/6 (tunneling)—parse headers and find layer 4 protocol
- IP layer 4 protocol/next header extraction

- Stop parsing on unrecognized next header/protocol
- IPv4 support
  - IPv4 source and destination addresses
  - 8-bit IPv4 type of service
  - IP layer 4 protocol / next header support
    - IPV4
    - IPV4 Fragment. Parser stops after a fragment is found
    - TCP/UDP
- IPv6 support
  - The first 4 bytes of the IPv6 source address extraction
  - The first 4 bytes of the IPv6 destination address extraction
  - IPv6 source address hash for pseudo header calculation
  - IPv6 destination address hash for pseudo header calculation
  - 8-bit IPv6 traffic class field extraction
  - Payload length field extraction
  - IP layer 4 protocol/next header support
    - IPV6
    - IPV6 fragment. Parser stops after a fragment is found
    - IPV6 route
    - IPV6 hop/destination
    - TCP/UDP
- L4 (TCP/UDP) support
  - Extraction of 16-bit source port number extraction
  - Extraction of 16-bit destination port number extraction
  - TCP checksum calculation (including pseudo header)
  - UDP checksum calculation if the checksum field is not zero (including pseudo header)

### 14.6.5.2 Receive Queue Filer

The receive queue filer receives protocol header properties extracted from the incoming frame by the eTSEC frame parse engine. A property is defined to be a field extracted from a packet header, such as a TCP port number or VLAN identifier. As soon as the last identifiable header has been recognized, the filer commences searching the receive queue filer table, comparing properties in the table against properties extracted from the frame. This table is illustrated in Figure 14-148. Software populates the table with property values, stored to the RQPROP field, and indicates how to match and interpret the properties by setting flags in the RQCTRL field. The eTSEC memory map provides access to these fields by way of an address register (RQFAR) and two porthole registers (RQFCR and RQFPR).

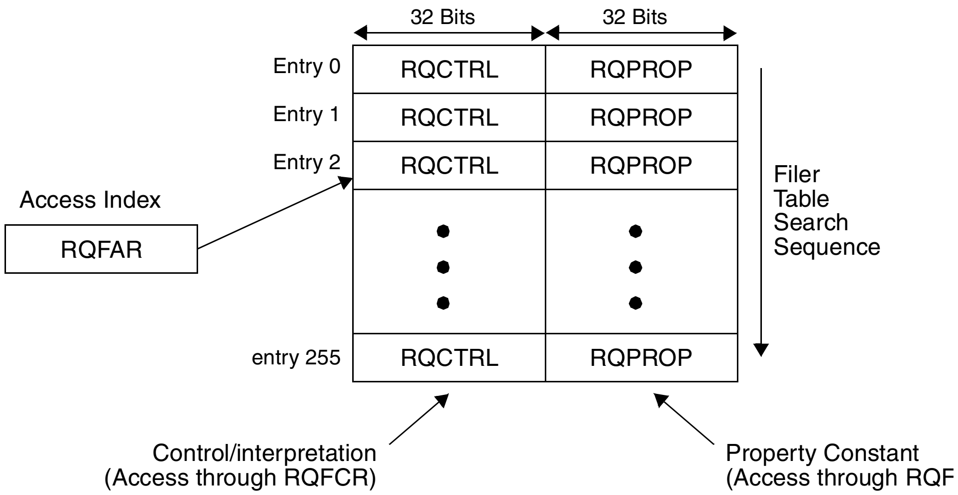


**Figure 14-148. Structure of the Receive Queue Filer Table**

### 14.6.5.2.1 Filing Rules

Unless the filer is disabled, every received frame from the Ethernet MAC or FIFO interface initiates a search of the receive queue filer table, starting at entry 0. The table search is terminated as soon as an entry is found whose contents match a property of the frame. Accordingly, software must guarantee that at least one entry results in a match—even if only to set a default receive queue index.

Since eTSEC searches the table at a rate of two entries every system clock cycle, all 256 entries can be searched in the time taken to receive a 64-byte Ethernet frame.

Each entry of the receive queue filer table specifies a simple match rule for determining how to process the received frame. The elements of a filing rule, expressed in the RQCTRL and RQPROP fields, are summarized as follows:

- The PID field in RQCTRL identifies what property is being matched against RQPROP. The eTSEC supports 16 properties, some of which are different portions of the same header field. Reserved or unused bits in RQPROP are read as zero. See Section 14.5.3.3.8, “Receive Queue Filer Table Property Register (RQFPR),” on page 14-65 for a list of all properties and their associated PID values.
- The Q field in RQCTRL identifies which one of 64 virtual receive queues the frame should be filed to (sent through DMA) in the event of a filing rule match that accepts the frame. The physical RxBD ring this queue maps to is controlled by the RCTRL[FSQEN] bit. If RCTRL[FSQEN] = 0, the three least significant bits of the Q field indicate which physical RxBD ring hosts the queue. If RCTRL[FSQEN] = 1, RxBD ring 0 hosts all receive queues, but the RxFCB[RQ] field allows software to distinguish queues by ID. In all cases if Q maps to a RxBD ring that is not currently enabled, the frame is discarded with an IEVENT[FIQ] error.
- The REJ field in RQCTRL controls whether the frame is to be rejected (REJ = 1) or filed (REJ = 0) upon a filing rule match. Rejected frames occupy Rx FIFO space, but do not consume memory bus cycles.
- The CMP field in RQCTRL determines how property PID is compared against RQPROP. Equality, inequality, greater-or-equal, and less-than compares are available.

- The AND field in RQCTRL allows more than one comparison in a sequence to be chained together as a Boolean AND condition. Setting AND = 1 defers evaluation of the rule until the next entry has been matched, which may, in turn, have AND set. If any comparison involving AND = 1 fails, the entire chained sequence fails. A typical use for AND is to combine a pair of comparisons in a range match; the first such entry has AND = 1, the second has AND = 0 and its values of Q and REJ take effect.
- The CLE field in RQCTRL offers a way to bracket a set of consecutive—perhaps related—rules into a rule cluster. A cluster must be preceded by a guard rule, which simply determines whether the cluster rules can be evaluated. If the guard rule succeeds and its last entry has both CLE = 1 and AND = 1, the cluster rules that follow are enabled. The cluster ends at the first entry where CLE = 1 and AND = 0, which may also belong to a rule that files or rejects a frame. If the guard rule fails, all rules in the cluster are skipped, including mask_register assignments. Clusters must not be nested.
- The GPI field offers the user the ability to interrupt the core upon matching a rule that causes a frame to be filed to memory. Once the last RxBD corresponding to that frame is written to memory, the IEVENT[FGPI] event will be asserted. This bit will be set regardless of any interrupt coalescing that may be set.

#### 14.6.5.2.2 Comparing Properties with Bit Masks

By default, extracted properties are compared arithmetically according to the CMP field in each RQCTRL word. This permits point value matches in each table entry, and range checks across a pair of table entries combined with the AND attribute in RQCTRL. However, inspection of the parse flags, Ethernet preamble, and IP addresses typically requires “don’t care” bit fields in the properties to be cleared as part of the comparison. The eTSEC provides a dedicated 32-bit register, known as the mask_register, for performing such masking operations. At the start of each table search by the filer, mask_register is reset to 0xFFFF_FFFF, which ensures that no masking occurs.

Filer rules may be configured to assign specific bit patterns to mask_register. Such rules can be configured to either match always (useful for implementing a default rule and specifying an associated receive queue), or fail always (which prevents termination of the filer table search). Once mask_register has been assigned, it retains its value until it is reassigned or the table search terminates. All properties are non-destructively bit-wise ANDed with mask_register prior to comparison in subsequent rules, which allows an entire cluster of rules to make use of a common mask. Individual masks for specific rules can also be created simply by combining a mask_register assignment (match always form) with a regular rule using the AND attribute.

To create a mask_register assignment rule, it is necessary to select PID = 0 in RQCTRL, and choose CMP such that the rule either matches (CMP = 01) or fails (CMP = 11). In this entry, RQPROP is then considered to be the assigned bit vector.

### 14.6.5.2.3 Special-Case Rules

It is frequently useful to create rules that are guaranteed to succeed or fail, specifically to enforce a default filing decision or act as null entries. Suggested constructions for such rules are shown in Table 14-161.

Table 14-161. Special Filer Rules

| Rule Description | RQCTRL Fields | | | | | | RQPROP Word | RQCTRL Word[1] |
|---|---|---|---|---|---|---|---|---|
| | CLE | REJ | AND | Q | CMP | PID | | |
| Default file—Always file frame to ring Q | 0 | 0 | 0 | Q | 01 | 0000 | 0x0000_0000 | 0x0000_*qq*20 |
| Default reject—Always discard frame | 0 | 1 | 0 | 000_000 | 01 | 0000 | 0x0000_0000 | 0x0000_0120 |
| Empty rule in AND—Always matches | 0/1[2] | 0 | 1 | 000_000 | 01 | 0000 | 0xFFFF_FFFF | 0x0000_00A0 |
| Empty rule in rule set—Always fails | 0/1[3] | 0 | 0 | 000_000 | 11 | 0000 | 0xFFFF_FFFF | 0x0000_0060 |

[1] Hexadecimal digits *qq* denotes field Q shifted left 2 bits.

[2] Set CLE = 1 if the empty rule guards a cluster.

[3] Set CLE = 1 if the empty rule occurs at the end of a cluster.

### 14.6.5.2.4 Filer Interrupt Events

The filer can produce three interrupt events in IEVENT. Event FIR indicates an error condition where the filer was unable to provide a definite result, either because no rule in the table succeeded, or because frames arrived too rapidly to complete searching of the table. Event FIQ indicates that the filer accepted a frame to a RxBD ring that was not enabled in RQCTRL (this can also occur if the filer is disabled, but RxBD ring 0—default queue or FSQEN mode queue—is not enabled). FIQ is also asserted in the case where no rule in the entire table succeeded. The various combinations of these interrupt events and their interpretation appear in Table 14-162.

Table 14-162. Receive Queue Filer Interrupt Events

| IEVENT[FIR] | IEVENT[FIQ] | Description |
|---|---|---|
| 0 | 0 | No error. The filer successfully rejected or filed a frame. |
| 0 | 1 | Illegal queue error. The filer accepted a frame to a RxBD ring that is disabled (including ring 0 if filing is disabled). |
| 1 | 0 | Partial search error. The filer did not have sufficient time to complete its search of the filer table. |
| 1 | 1 | No matching rule error. The filer searched all 256 entries of the filer table without finding a rule that succeeds. |

A functional interrupt is provided via use of the general purpose interrupt (GPI) bit in the filer table. When a property matches the value in the RQPROP entry at this index, and REJ = 0 and AND = 0, the filer will set IEVENT[FGPI] when the corresponding receive frame is written to memory. This allows the user to set up a filer rule where the core will be interrupted upon the reception of ‘special’ frames.

If the timer is enabled (TMR_CTRL[TE] = 1), then the interrupt dedicated for timer events (in addition to the usual receive, transmit and error interrupts) will be asserted.

### 14.6.5.2.5 Setting Up the Receive Queue Filer Table

The eTSEC frame parser always provides values for all properties, even where the relevant headers are not available. In the latter case, the filer is given default properties that can be used to avoid conflict with normal, defined property values. Accordingly, the rules in the filer table can be partitioned into rule sets such that if all rules in a given set fail (due to headers being unavailable), lower priority rule sets can be subsequently searched until either a rule set provides a match or a single default—catch-all—rule specifies a definite receive queue. For example, an IEEE 802.1p priority rule set may be followed by an IP TOS rule set, followed by a default rule; thus, if no VLAN tag appears in the received frame, the TOS rules are checked, or the default is activated should no IP header be present.

The rule cluster feature is used to conditionalize evaluation of rule sets. Typically, this avoids evaluating rules based on properties that may not be valid or relevant to the filing or filtering decision. For example, TCP-related rules might be clustered behind a guard rule that checks that a TCP header has appeared and the IP address matches our home address. Property 1—the parse flags property—is provided specifically to check the characteristics of the received frame and the parser error status. The mask_register is typically assigned beforehand to extract specific flags, in which case care should be taken that mask_register be reassigned an appropriate mask vector for following comparisons.

In many cases it is possible to write the entire filer table before using eTSEC, as the rule set is static. However, dynamic rule updates can be supported by pre-allocating partially instantiated rule sets, which software rewrites as necessary. Rules that are not instantiated should be composed of empty entries, as indicated in Table 14-161. In many cases empty entries can be overwritten by software without stopping eTSEC's receive function.

### 14.6.5.2.6 Filer Example—802.1p Priority Filing

This example, shown in Table 14-163, illustrates how to file frames according to layer 2 802.1p priority. This matches against property 1001, comparing each specific priority level in order to associate them with a RxBD ring index. Note that if a VLAN tag does not appear in the frame, the parser passes priority 0 to the filer, which always matches the rule at entry 7 and terminate the table search.

**Table 14-163. Filer Table Example—802.1p Priority Filing**

| Table Entry | RQCTRL Fields | | | | | | RQPROP | Comment | RQCTRL Word |
|---|---|---|---|---|---|---|---|---|---|
| | CLE | REJ | AND | Q | CMP | PID | | | |
| 0 | 0 | 0 | 0 | 000_000 | 00 | 1001 | 0x0000_0007 | File priority 7 to ring 0 | 0x0000_0009 |
| 1 | 0 | 0 | 0 | 000_001 | 00 | 1001 | 0x0000_0006 | File priority 6 to ring 1 | 0x0000_0409 |
| 2 | 0 | 0 | 0 | 000_010 | 00 | 1001 | 0x0000_0005 | File priority 5 to ring 2 | 0x0000_0809 |
| 3 | 0 | 0 | 0 | 000_011 | 00 | 1001 | 0x0000_0004 | File priority 4 to ring 3 | 0x0000_0C09 |
| 4 | 0 | 0 | 0 | 000_100 | 00 | 1001 | 0x0000_0003 | File priority 3 to ring 4 | 0x0000_1009 |
| 5 | 0 | 0 | 0 | 000_101 | 00 | 1001 | 0x0000_0002 | File priority 2 to ring 5 | 0x0000_1409 |

Table 14-163. Filer Table Example—802.1p Priority Filing (continued)

| Table Entry | RQCTRL Fields | | | | | | RQPROP | Comment | RQCTRL Word |
|---|---|---|---|---|---|---|---|---|---|
| | CLE | REJ | AND | Q | CMP | PID | | | |
| 6 | 0 | 0 | 0 | 000_110 | 00 | 1001 | 0x0000_0001 | File priority 1 to ring 6 | 0x0000_1809 |
| 7 | 0 | 0 | 0 | 000_111 | 00 | 1001 | 0x0000_0000 | File undefined 802.1p or priority 0 to ring 7—Default always matches | 0x0000_1C09 |

### 14.6.5.2.7 Filer Example—IP Diff-Serv Code Points Filing

This example demonstrates use of rule priority for determining class selector codepoints (RFC 2474) from the IP TOS property. An example filer table is shown in Table 14-164. The example relies on the fact that the first rule matched terminates the search, hence successively lower Diff-Serv codepoint ranges can be compared in each step until the default (zero or greater) range is reached. By default, property 1010 (IP TOS) takes the value 0x00 if no IP headers were recognized, therefore the table search always terminates.

Table 14-164. Filer Table Example—IP Diff-Serv Code Points Filing

| Table Entry | RQCTRL Fields | | | | | | RQPROP | Comment | RQCTRL Word |
|---|---|---|---|---|---|---|---|---|---|
| | CLE | REJ | AND | Q | CMP | PID | | | |
| 0 | 0 | 0 | 0 | 001_000 | 01 | 1010 | 0x0000_00E0 | File class 7 to queue 8 (TOS >= 0xE0) | 0x0000_202A |
| 1 | 0 | 0 | 0 | 001_001 | 01 | 1010 | 0x0000_00C0 | File class 6 to queue 9 (TOS >= 0xC0) | 0x0000_242A |
| 2 | 0 | 0 | 0 | 001_010 | 01 | 1010 | 0x0000_00A0 | File class 5 to queue 10 (TOS >= 0xA0) | 0x0000_282A |
| 3 | 0 | 0 | 0 | 001_011 | 01 | 1010 | 0x0000_0080 | File class 4 to queue 11 (TOS >= 0x80) | 0x0000_2C2A |
| 4 | 0 | 0 | 0 | 000_100 | 01 | 1010 | 0x0000_0060 | File class 3 to queue 4 (TOS >= 0x60) | 0x0000_102A |
| 5 | 0 | 0 | 0 | 001_100 | 01 | 1010 | 0x0000_0040 | File class 2 to queue 12 (TOS >= 0x40) | 0x0000_302A |
| 6 | 0 | 0 | 0 | 010_100 | 01 | 1010 | 0x0000_0020 | File class 1 to queue 20 (TOS >= 0x20) | 0x0000_502A |
| 7 | 0 | 0 | 0 | 011_100 | 01 | 1010 | 0x0000_0000 | File class 0 to queue 28 (TOS >= 0x00) or file to ring 4 by default | 0x0000_702A |

### 14.6.5.2.8 Filer Example—TCP and UDP Port Filing

This example demonstrates rule clusters and AND-combined entries for filing packets based on transport protocol and well-known port numbers in a termination application. An example filer table is shown in Table 14-165. The example contains two clusters; the first is entered only for TCP packets, the second is entered only for UDP packets. A default filing rule catches the case where neither TCP nor UDP headers are found. Each cluster compares source port number (property 1111) against a list of server ports, and files the packets accordingly. Note that entries 1 and 2 form an AND rule for checking that the port number >= 20 and port number < 22. Entries 4 and 5 are initially set up to always fail (zero port number), and thus comprise empty entries that can be used at a later time.

Table 14-165. Filer Table Example—TCP and UDP Port Filing

| Table Entry | RQCTRL Fields | | | | | | RQPROP | Comment | RQCTRL Word |
|---|---|---|---|---|---|---|---|---|---|
| | CLE | REJ | AND | Q | CMP | PID | | | |
| 0 | 1 | 0 | 1 | 000_000 | 00 | 1011 | 0x0000_0006 | Enter cluster if layer 4 is TCP | 0x0000_028B |
| 1 | 0 | 0 | 1 | 000_000 | 01 | 1111 | 0x0000_0014 | AND rule—FTP from TCP ports 20 and 21: file to ring 2 | 0x0000_00AF |
| 2 | 0 | 0 | 0 | 000_010 | 11 | 1111 | 0x0000_0016 | | 0x0000_086F |
| 3 | 0 | 0 | 0 | 000_011 | 00 | 1111 | 0x0000_0017 | telnet from TCP port 23: file to ring 3 | 0x0000_0C0F |
| 4 | 0 | 0 | 0 | 000_000 | 00 | 1111 | 0x0000_0000 | *empty entry reserved for future use* | 0x0000_000F |
| 5 | 0 | 0 | 0 | 000_000 | 00 | 1111 | 0x0000_0000 | *empty entry reserved for future use* | 0x0000_000F |
| 6 | 1 | 0 | 0 | 000_001 | 01 | 0000 | 0x0000_0000 | end cluster; default TCP: file to ring 1 | 0x0000_0620 |
| 7 | 1 | 0 | 1 | 000_000 | 00 | 1011 | 0x0000_0011 | Enter cluster if layer 4 is UDP | 0x0000_028B |
| 8 | 0 | 0 | 0 | 000_101 | 00 | 1111 | 0x0000_0801 | NFS from UDP port 2049 | 0x0000_140F |
| 9 | 0 | 0 | 0 | 000_111 | 00 | 1111 | 0x0000_0208 | Route from UDP port 520 | 0x0000_000F |
| 10 | 0 | 0 | 0 | 000_110 | 00 | 1111 | 0x0000_0045 | TFTP from UDP port 69 | 0x0000_180F |
| 11 | 1 | 0 | 0 | 000_100 | 01 | 0000 | 0x0000_0000 | End cluster; default UDP: file to ring 4 | 0x0000_1220 |
| 12 | 0 | 0 | 0 | 000_000 | 01 | 0000 | 0x0000_0000 | By default, file to ring 0 | 0x0000_0020 |

### 14.6.5.3 Transmission Scheduling

Each eTSEC can maintain multiple TxBD rings (or transmission queues) to satisfy QoS requirements. The ability to choose from a number of transmission streams dynamically is especially important during periods of network congestion. Certain application such as voice and video streaming are delay sensitive, but loss insensitive. For instance, VoIP applications require little bandwidth, but are highly sensitive to latency. Conversely, FTP or SMTP protocols are delay insensitive, but loss sensitive.

eTSEC has a transmission scheduler that implements a programmable QoS regime. The scheduler is responsible for choosing which of the prefetched TxBDs shall be processed next, and accordingly issuing DMA requests to service the data stream described by the chosen BD(s). The scheduler cycle is one of:

1. decide on a TxBD queue,
2. transmit exactly one frame from that queue, and
3. return to deciding on another queue, in step 1.

If TCTRL[TXSCHED] is set to 00, no transmission scheduling occurs, and only TxBD ring 0 is polled for new data to transmit, with DMACTRL controlling waiting or polling. TCTRL[TXSCHED], if not zero, can be programmed to invoke one of two scheduling algorithms, namely priority-based queuing (PBQ), and modified weighted round-robin queuing (MWRR). In all cases where TCTRL[TXSCHED] is not zero, the scheduler can choose from among 1 to 8 TxBD rings per eTSEC, with individual rings being enabled by the setting of TQUEUE[EN0–EN7] bits. For example, TxBD rings 3, 4, and 7 may be enabled for scheduling by setting EN3, EN4, and EN7, and clearing all other EN bits.

#### 14.6.5.3.1 Priority-Based Queuing (PBQ)

PBQ is the simplest scheduler decision policy. The enabled TxBD rings are assigned a priority value based on their index. Rings with a lower index have precedence over rings with higher indices, with priority assessed on a frame-by-frame basis. For example, frames in TxBD ring 0 have higher priority than frames in TxBD ring 1, and frames in TxBD ring 1 have higher priority than frames in TxBD ring 2, and so on.

The scheduling decision is then achieved as follows:

```
loop
    # start or S/W clear of TSATn
    ring = 0;
    while ring <= 7 loop
     if enabled(ring) and not ring_empty(ring) then
          transmit_frame(ring);
          ring = 0;
     else
          ring = ring + 1;
     endif
    endloop
endloop
```

#### 14.6.5.3.2 Modified Weighted Round-Robin Queuing (MWRR)

eTSEC implements a modified weighted round-robin scheduling algorithm across all enabled TxBD rings when TCTRL[TXSCHED] = 10. In MWRR, the weights in the TR03WT and TR47WT registers determine the ideal size of each transmit slot, as measured in multiples of 64 bytes. Thus, to set a transmit slot of 512 bytes, a weight of 512/64 or 8 needs to be set for the ring. In this mode TxBD rings 1–7 are selected in round-robin fashion, whereas TxBD ring 0, if enabled with ready data for transmission, is always selected in between other rings so as to expedite transmission from ring 0.

The scheduling decision is then achieved as follows:

```
for ring = 1..7 and enabled(ring) loop
    credit[ring] = 0;
endloop
for ring = 1..7 and enabled(ring) loop
    if not ring_empty(0) then
     credit[0] = credit[0] + weight[0];
     while credit[0] > 0 loop
          transmit_frame(0);
          credit[0] = credit[0] - frame_size;
          if ring_empty(0) then
           credit[0] = 0;
          endif
     endloop
    endif
    if not ring_empty(ring) then
     credit[ring] = credit[ring] + weight[ring];
    endif
    while credit[ring] > 0 loop
     transmit_frame(ring);
     credit[ring] = credit[ring] - frame_size;
     if ring_empty(ring) then
          credit[ring] = 0;
     endif
```

```
        endloop
    endloop
```

The algorithm checks registers TQUEUE[EN0–EN7] for `enabled()`, TSTAT[THLT0–THLT7] for `ring_empty()`, and TR*x*WT for `weight()`. For TxBD ring *k*, having a weight WT*k*, the long term average throughput for that ring is:

rate of queue[k] (K = 1 to 7) = (available bandwidth) * WT*k*/(sum(WT*i*) + 6WT0)

rate of queue(0) = (available bandwidth) * 7 * WT0/(sum(WT*i*) + 6WT0)

where *i* = 0 to 7

## 14.6.6 Lossless Flow Control

The eTSEC DMA subsystem is designed to be able to support simultaneous receive and transmit traffic at gigabit line rates. If the host memory has sufficient bandwidth to support such line rates, then the principle cause of overflow on receive traffic is due to a lack of Rx BDs. Thus, the long term receive throughput is determined by the rate at which software can process receive traffic. If a user desires to prevent dropped packets, they can inform the far-end link to stop transmission while the software processing catches up with the backlog.

To avoid overflow in the latter case, back pressure must be applied to the far-end transmitter before the Rx descriptor controller encounters a non-empty BD and halts with a BSY error. As there is lag between application of back-pressure and response of the far-end, the pause request must be issued while there are still BDs free in the ring. In the traditional eTSEC descriptor ring programming model, there is no way for hardware to know how many free BDs are available, so software must initiate any pause requests required during operation. If software is backlogged, the request may be not be issued in time to prevent BSY errors. To allow the eTSEC to generate the pause request automatically, additional information (a pointer the last free BD and ring length) is required.

### 14.6.6.1 Back Pressure Determination through Free Buffers

Ultimately, the rate of data reception is determined by how quickly software can release buffers back into the receive ring(s). Each time a buffer is freed, the associated BD has its empty bit set and hardware is free to consume both. Thus the number of free BDs in a given Rx ring indicates how close hardware is to the end of that ring. To prevent data loss, back pressure should be applied when the number of free BDs drops below some critical level. The number of BDs that can be consumed by an incoming packet stream while back-pressure takes effect is determined by several factors, such as: receive traffic profile, transmit traffic profile, Rx buffer size, physical transmission time between eTSEC and far-end device and intra-device latency. Theoretically, the worst case is as follows:

$$\mathrm{FreeBDsRequired} = \frac{\mathrm{MaxFrameSize}}{\mathrm{MinFrameSize} + \mathrm{IFG}} + \frac{\mathrm{MaxFrameSize}}{\mathrm{RxBufferSize}} + \mathrm{LinkDelay}$$

This case comes about when:

- The eTSEC has just started transmitting a large frame and thus cannot send out a pause frame
- Upon reception of the pause request the far-end has just started transmission of a large frame

- The eTSEC receives a burst of short frames with minimum inter-frame-gap (96bit times for ethernet)

Once the user has determined the worst case scenario for their application, they program the required free BD threshold into the eTSEC (through RQPRM[PBTHR]). Since different BD rings may have different sizes and expected packet arrival rates, a separate threshold is provided for each active ring. It is recommended that a threshold of at least fourBDs is the practical minimum for gigabit ethernet links.

For the Rx descriptor controller to determine the number of free BDs remaining in the ring, it needs to know the following:

1. The location of the current BD being used by hardware
2. The location of the last BD that was released (freed) by software
3. The length of the Rx BD ring.

For each active ring, the current BD pointer (RBPTR*n*) is maintained by the eTSEC. Software knows both the size of the Rx ring and the location of the last freed BD. By providing the eTSEC with those values (through RQPRM[LEN] and RFBPTR respectively) the eTSEC always know how many receive buffers are available to be consumed by incoming data.

The number of guaranteed free BDs in the ring is then determined by:

When RFBPTRn < RBPTR*n*

$$\text{FreeBDs} = \text{RQPRM}n[\text{LEN}] - \text{RBPTR}n + \text{RFBPTR}n$$

When RFBPTRn > RBPTR*n*

$$\text{FreeBDs} = \text{RFBPTR}n - \text{RBPTR}n$$

When RBPTR*n* = RFBPTR*n* the number of free BDs in the ring is either one (since RFBPTR*n* points to a free BD) or equal to the ring length. Since the BD pointed to by RBPTR*n* may be either in use or about to be used, it is not considered in the free BD count. To resolve the case where the two pointers collide, the following logic applies:

If RBASE*n* was updated and thus initializes both RBPTR*n* and RFBPTR*n*, the ring is deemed empty.

If RFBPTR*n* is updated by a software write and matches RBPTR*n*, the ring is deemed empty.

If HW updates RBPTR*n* and the result matches RFBPTR*n*, the ring is deemed to have one BD remaining. Upon writing this BD back to memory (indicating the buffer is occupied) the ring is deemed to be full.

**Important.** There is a possibility that if software is severely backlogged in updating RFBPTR*n*, the hardware could wrap around the ring entirely, consume exactly the remaining number of BDs and not halt with a BSY error. If software then increments RFBPTR*n* to the next address (thereby equalling RBPTR*n*), the hardware assumes the ring is now empty (when in fact there is only a single BD freed up). This results in the hardware failing to maintain back pressure on the far end. Upon software incrementing RFBPTR*n* a subsequent time, the wrap condition is successfully detected and hardware recognizes a nearly full ring (rather than a nearly empty one). Since software can increment RFBPTR*n* by any amount, it is not possible for hardware to determine in this case whether the user has cleared the entire ring or just one BD. Users can eliminate the possibility of this condition occurring by ensuring that RFBPTR*n* is incremented by at least two BDs each time (that is, clear at least two buffers whenever the RxBD unload routine is called).

Once the eTSEC determines that this threshold has been reached, back pressure is applied accordingly. The type of back pressure that is applied varies according to the physical interface that is used.

- **Half duplex Ethernet:** No support in this mode.
- **Full duplex Ethernet:** An IEEE 802.3 PAUSE frame (see sect. 14.6.3.9/14-170) is issued as if the TCTRL[TFC_PAUSE] bit was set. An internal counter tracks the time the far end controller is expected to remain in pause (based on the setting of PTV[PT]). When that counter reaches half the value of PTV[PT], the eTSEC reissues a pause frame if the free BD calculation for any ring is below the threshold for that ring. For example, if PTV[PT] is set to 10 quanta, a pause frame is re-issued when five quanta have elapsed if the free BD threshold is still not met. A practical minimum for PTV[PT] of 4 quanta is recommended.
- **FIFO packet interface:** Link layer flow control is asserted through use of the RFC signal (CRS pin). Flow control is asserted for the entire time that free BD threshold is not met. The same mechanism is used for both GMII-style and encoded packet modes.

### 14.6.6.2 Software Use of Hardware-Initiated Back Pressure

#### 14.6.6.2.1 Initialization

Software configures RBASE*n* and RQPRM*n*[LEN] according to the parameters for that ring. Then the number of free BDs that are required to prevent the eTSEC from automatically asserting flow control are programmed in RQPRM[FBTHR]. The receiver is then enabled.

Note: the act of programming RBASE*n* initializes RFBPTR*n* to the start of the of the ring. When the ring is in this initial empty state, there is no concept of a last freed BD. In this case, the calculated number of free BDs is the size of the ring. Since the BD that the hardware is currently pointing to is to be considered in-use, the free BD count is actually one higher than the total available. As soon as the hardware consumes a BD (by writing it back to memory), RBPTR*n* advances and the free BD count reflects the correct number of available free BDs.

#### 14.6.6.2.2 Operation

As software frees BDs from the ring, it writes the physical address of the BD just freed to RFBPTR*n*. The eTSEC asserts flow control if the distance (using modulo arithmetic) between RBPTR*n* and RFBPTR*n* is < RQPRM*n*[FBTHR]. In multi-ring operation, if the free BD count of **any** active ring drops below the threshold for that ring, flow control is asserted. Once enough BDs are freed for **all** active rings to meet their respective free BD thresholds, application of back pressure cases.

Note: The eTSEC does not issue an exit pause frame (that is, pause frame with PTV of 0x0000) once all active rings have sufficient BDs. Instead, it waits for the far-end pause timer to expire and start re-transmission.

### 14.6.7 Hardware Assist for IEEE Std. 1588-CompatibleTimestamping

There is a push in industrial control applications to use Ethernet as the principal link layer for communications. This requires Ethernet to be used for both data transfer and real-time control. For real-time systems, each node is required to be synchronized to a master clock. The precision of this clock is dictated by the application, but generally needs to be of the order of <1uSec for high-speed machinery (for example, printing presses).

IEEE 1588 [1588] specifies a mechanism for synchronizing multiple nodes to a master clock. Support for 1588 can be done entirely in software running on a host CPU, but applications that require sub 10uSec accuracy will need hardware support for accurate timestamping of incoming packets.

The eTSEC includes a new timer clock module to support the IEEE Std. 1588 timer standard. The following sections describe the features, programming model, and implementation information.

**NOTE**

> IEEE 1588 timestamping is not supported in conjunction with the SGMII 10/100 interface mode.

#### 14.6.7.1 Features

- 64-bit free running timer running from an external oscillator or internal clock
- Programmable timer oscillator clock selection
- Self-correcting precision timer with nano-second resolution
- Time stamp all incoming packets inline
  - Maskable interrupts on received PTP packet's filer rule match
- Time stamp transmit packets when instructed in the TxFCB
  - Maskable interrupts on transmit timestamp capture
- Two Tx time stamp registers per eTSEC with 16-bit tag for each of them to support burst mode.
- Time stamp capture on two general-purpose external triggers
  - Maskable interrupts on GPIO timestamp trigger
  - Programmable polarity of external trigger (GPIO) edge
- Two 64-bit alarm (future time) registers for future time comparison
  - Maskable interrupts on alarm
- Three programmable timer output pulse period phase aligned with 1588 timer clock
  - Maskable interrupts associated with each pulse
- Separate maskable timer interrupt event register
- Recognition of incoming PTP packet through filer rule match
- Phase aligned adjustable (divide by N) clock output
- Supports all Ethernet modes supported by the eTSEC, including full- and half-duplex modes
- Supports both master and slave modes
- Supports timestamp of nano-second resolution

#### 14.6.7.2 Timer Logic Overview

The 1588 timer module can be partitioned into four different sub-modules as shown in Figure 14-149.

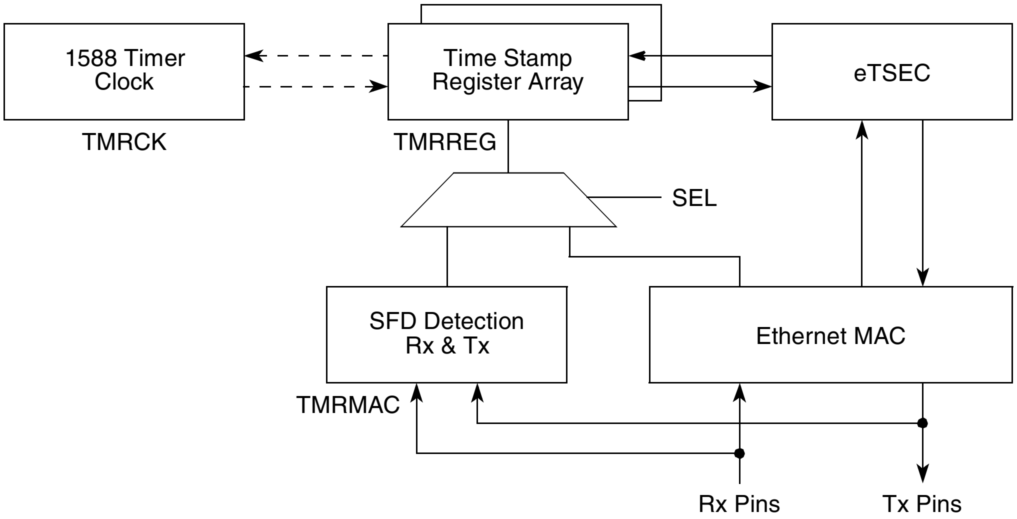


Figure 14-149. 1588 Timer Design Partition

#### 14.6.7.3 Time-Stamp Insertion on the Received Packets

Every incoming packet's 8-byte time stamp is inserted into the packet data buffer as padding alignment bytes. Time-stamp insertion into the data buffer requires RCTRL[PAL] to be set to a value greater than or equal to 8 and the control bit RCTRL[TS] bit to be set.

##### 14.6.7.3.1 Timestamp Point

The required timestamp point, as specified in the IEEE 1588 Specification Sep-2004 (IEC 61588 First Edition), is shown in Figure 14-150. From this, it is clear that the end of the SFD is the critical point in the MII data stream.

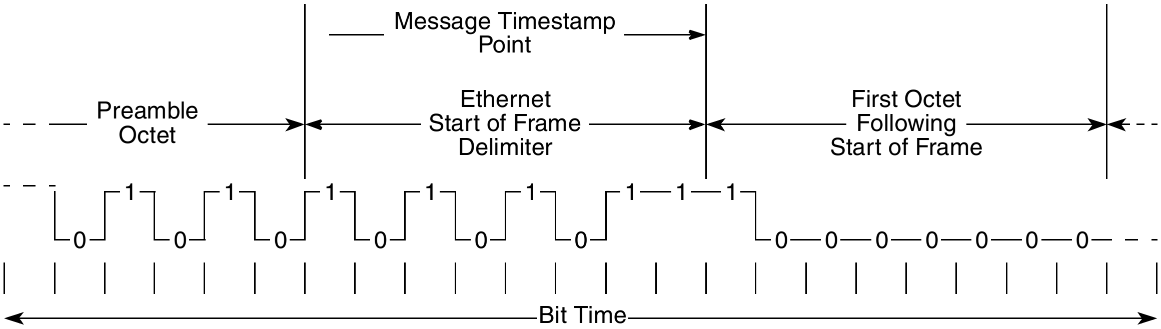

Message Timestamp Point
Preamble Octet
Ethernet Start of Frame Delimiter
First Octet Following Start of Frame
1
0
Bit Time

Figure 14-150. Ethernet Sampling Points for 1588

The sample point coincides with the cycle after the SFD (Start of Frame Delimiter) detection by the MAC. For received frames, this will be at least 4 bit times (MII) or 8 bit times (GMII) after the message timestamp point specified in [1588]. For transmission, the eTSEC sample point precedes the sample point specified in [1588] by at least 4-bit times (MII) or 8-bit times (GMII). For a particular mode, the eTSEC sample point is a consistent number of bit times relative to the SFD detection. Thus, the offset from the [1558] specified sample point can be accounted for in the PTP software implementation.

### 14.6.7.4 PTP Packet Parsing

PTP packets are typically embedded within a UDP payload with special IP source and destination address and special source and destination ports numbers. Special fields of interest of a PTP packet are listed in Table 14-166.

**Table 14-166. PTP Payload Special Fields**

| Layer | Octet (Offset from the SFD) | Field | Value | eTSEC filer PID | Comments |
|---|---|---|---|---|---|
| Ethernet | 12-13 | Length/Packet | 0x0800 | ETY-RQPFR[PID=0111] | IPv4 |
| IP header | 22 | Time to live | 0x00 | RBIFX-choose an arbitrary extraction byte | Must be 0 |
| IP header | 23 | IP Protocol | 0x11 | L4P-RQPFR[PID=1011] | UDP |
| IP header | 26-29 | Source IP Address<br>IANA defines 4 multicast address for the PTP packet | | SIA-RQPFR[PID=1101] | |
| IP header | 30-33 | Destination IP Address<br>IANA defines 4 multicast address for the PTP packet | 224.0.1.129<br>224.0.1.130<br>224.0.1.131<br>224.0.1.132 | DIA-RQPFR[PID=1100] | DefaultPTPdomain<br>AlternatePTPdomain1<br>AlternatePTPdomain2<br>AlternatePTPdomain3 |
| UDP header | 34-35 | Source port number | | SPT-RQPFR[PID=1011] | |
| UDP header | 36-37 | Destination port number | 319<br>320 | DPT-RQPFR[PID=1011] | EventPort<br>GeneralPort |
| UDP data | 74 | Control | 0x0<br>0x1<br>0x2<br>0x3<br>0x4 | RBIFX-choose an arbitrary extraction byte | Sync<br>Delay_req<br>Follow_up<br>Delay_resp<br>Management |

A representation of the PTP packet is shown in Figure 14-151.

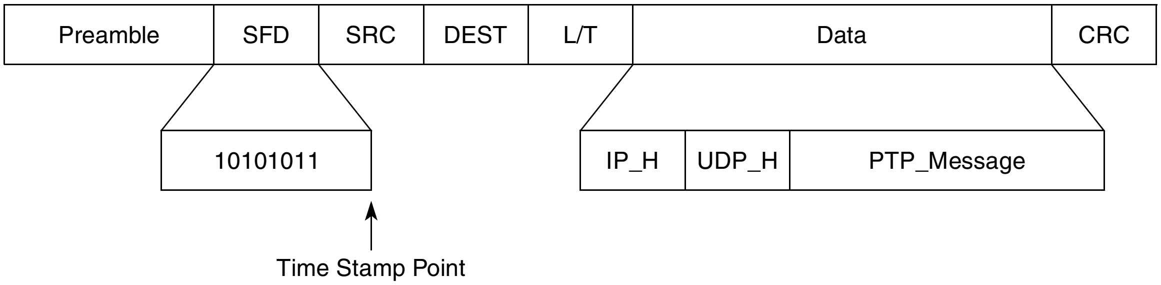


**Figure 14-151. PTP Packet Format**

#### 14.6.7.4.1 General Purpose Filer Rule

The eTSEC receive filer has been enhanced with the addition of a general-purpose event bit. This event bit can be used in conjunction with filing table rules to identify 1588 packets and indicate these packets by setting special timer status register bits (TMR_STAT). Additionally, 1588 packets can be easily identified by upper-layer software by using the filer to queue all PTP packets to one or more predefined virtual queues. See Section 14.6.5.2.1, “Filing Rules for further information.

### 14.6.7.5 Time-Stamp Insertion on Transmit Packets

Software has the option to write the time stamp of the transmitted frame to memory in the padding alignment bytes (PAL) located between the TxFCB and the frame data. It is required that a minimum of two TxBDs are used. The first points to the start of the 8 byte TxFCB. The second points to the start of frame data. In memory, the TxFCB, and at least the first 16 bytes of the TxPAL must be adjacent, i.e., located in continguous memory locations, as depicted in Figure 14-152.

The first TxBD[TOE] bit is set. When the TMR_CTRL[Record Time-stamp In PAL Enable] and TxFCB[PTP] bits are set, the timestamp is written to memory location TxBD[Data Buffer Pointer]+16.

The second TxBD’s Data Length must either contain the full frame length, or a value greater than the TxThreshold setting. Refer to Table 14-167. When time-stamps are inserted into the TxPAL, the TMR_TXTS*n*_H/L and TMR_TXTS*n*_ID registers still function normally.

#### 14.6.7.5.1 Interrupts

The TxPAL is updated with a time-stamp before closing the second TxBD. The TxBD[I] bit can be set for the second TxBD frame to cause an interrupt (via IEVENT[TXF]) after the time-stamp has been written to the TxPAL.

When time-stamps are inserted into the TxPAL, the TMR_TXTS*n*_H/L and TMR_TXTS*n*_ID registers still function normally. Therefore, the 1588 interrupt can be triggered by using the TMR_PEVENT register bits TXP1, and TXP2.

**Table 14-167. Time-Stamp Insertion Programming Requirements**

| Requirement | Behavior if requirement is not met |
|---|---|
| TMR_CTRL[RTPE]=1 | If TMR_CTRL[RTPE]=0, then no time-stamp is written to a TxPAL. |

Table 14-167. Time-Stamp Insertion Programming Requirements

| Requirement | Behavior if requirement is not met |
|---|---|
| TxBD[TOE]=1 | If TxBD[TOE]=0, then no time-stamp is written to a TxPAL. |
| First TxBD[Data Buffer Pointer] is 8-byte aligned | The time-stamp will be written to address First TxBD[Data Buffer Pointer] + 0x10 rounded down to the nearest 8-byte aligned address, except at 4K page boundaries, in which case the time-stamp may be invalid, and the Second TxBD close status will be lost. |
| First TxBD[Data Length]=8, 8 bytes for TxFCB | If L2 or frame data is included in the Length, the buffer immediately following the FCB is transmitted on the line and the frame data stored in memory will be overwritten with a time-stamp value after the frame is transmitted. |
| TxFCB[PTP]=1 | If TxBD[PTP]=0, then no time-stamp is written to a TxPAL. |
| The TxFCB is followed immediately by a minimum of 16 bytes for the TxPAL | The time-stamp will be written to address First TxBD[Data Buffer Pointer] + 0x10. |
| Second TxBD[Data Buffer Pointers] points to start of L2 or frame data | If there is only one TxBD used to transfer a PTP frame, then no time-stamp is written to a TxPAL. |
| Second TxBD[Data Length] >= FIFO_TX_THR or includes the entire frame | If this condition is not true, the time-stamp in TxPAL is invalid. |

Figure 14-152 depicts the buffer format requirements for time-stamp insertion on transmit packets.

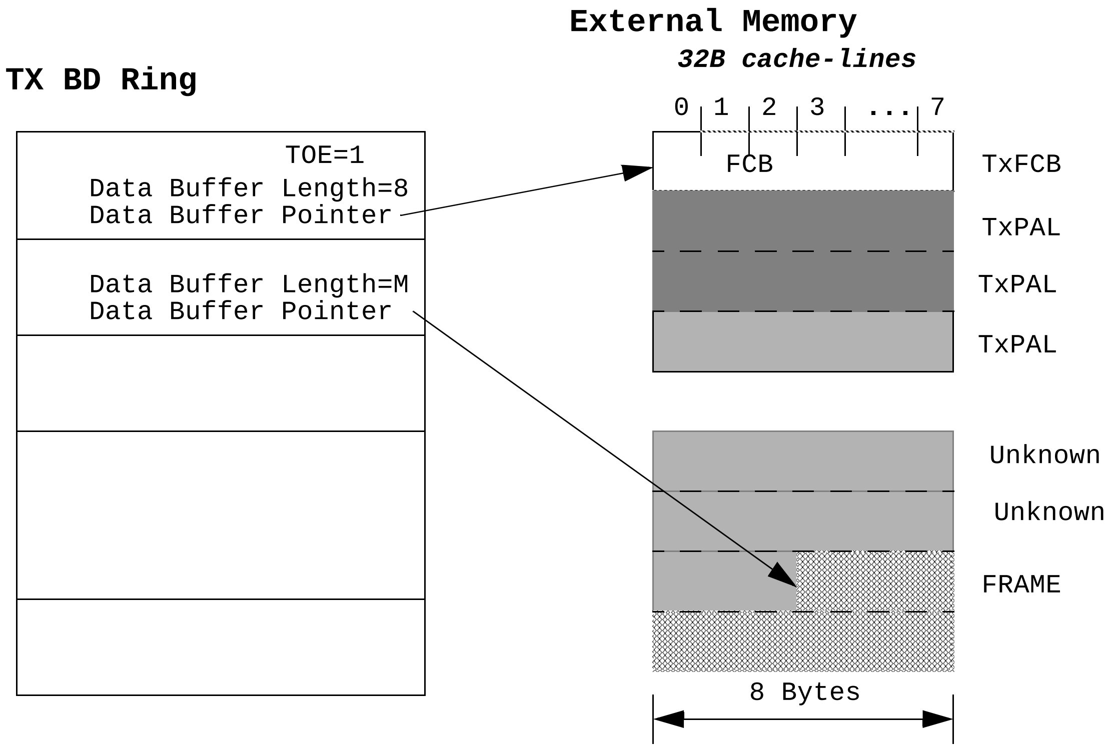


Figure 14-152. Buffer Format for Transmit Time-Stamp Insertion

#### 14.6.7.5.2 Error Condition

When an error is encountered after a PTP packet has begun to be processed, the time-stamp written to the TxPAL is zero. Subsequent frames may be flushed by eTSEC. There will be no time-stamp update to TxPAL for the subsequent flushed frames.

### 14.6.7.6 Tx PTP Packet Parsing

Software instructs the Tx packet to be timestamped via setting bit 15 in the TxFCB to mark a PTP packet. TxFCB[VLCTL] can be translated as the Tx PTP packet identification number. BD[TOE] must be set to enable transmit PTP packet time stamping. TxFCB[PTP] bit takes precedence over TxFCB[VLN] bit. It disables per packet VLAN tag insertion. On a PTP packet, a VLAN tag can be inserted from the DFVLAN register. The TxFCB for the PTP packet is shown in Figure 14-153.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Offset + 0 | VLN | IP | IP6 | TUP | UDP | CIP | CTU | NPH | | | | | | | | PTP |
| Offset + 2 | L4OS | | | | | | | | L3OS | | | | | | | |
| Offset + 4 | PHCS | | | | | | | | | | | | | | | |
| Offset + 6 | VLCTL/PTP_ID | | | | | | | | | | | | | | | |

**Figure 14-153. Transmit Frame Control Block**

The contents of the Tx FCB are defined in Table 14-168.

**Table 14-168. Tx Frame Control Block Description**

| Bytes | Bits | Name | Description |
|---|---|---|---|
| 0–1 | 0 | VLN | VLAN control word valid. This bit is ignored when the PTP bit is set. VLAN tag is read from the DFVLAN register if PTP=1.<br>0 Ignore VLCTL field.<br>1 If VLAN tag insertion is enabled for eTSEC, use the VLCTL field as the VLAN control word. |
| | 1 | IP | Layer 3 header is an IP header.<br>0 Ignore layer 3 and higher headers.<br>1 Assume that the layer 3 header is an IPv4 or IPv6 header, and take L3OS field as valid. |
| | 2 | IP6 | IP header is IP version 6. Valid only if IP = 1.<br>0 IP header version is 4.<br>1 IP header version is 6. |
| | 3 | TUP | Layer 4 header is a TCP or UDP header.<br>0 Do not process any layer 4 header.<br>1 Assume that the layer 4 header is either TCP or UDP (see UDP bit), and offload checksumming on the basis that the IP header has no extension headers. |
| | 4 | UDP | UDP protocol at layer 4.<br>0 Layer 4 protocol is either TCP (if TUP = 1) or undefined.<br>1 Layer 4 protocol is UDP if TUP = 1. |

Table 14-168. Tx Frame Control Block Description (continued)

| Bytes | Bits | Name | Description |
|---|---|---|---|
| 0–1 | 5 | CIP | Checksum IP header enable.<br>0 Do not generate an IP header checksum.<br>1 Generate an IPv4 header checksum. |
| | 6 | CTU | Checksum TCP or UDP header enable.<br>0 Do not generate a TCP or UDP header checksum. RFC 768 advises that UDP packets not requiring checksum validation should have their checksum field set to zero.<br>1 Generate a TCP header checksum if IP = 1 and TUP = 1 and UDP = 0. |
| | 7 | NPH | Disable calculation of TCP or UDP pseudo-header checksum. This bit should be set if IP options need to be consulted in forming the pseudo-header checksum, as eTSEC does not examine IP options or extension headers for TCP/IP offload on transmit.<br>0 Calculate TCP or UDP pseudo-header checksum as normal, assuming that the IP header has no options.<br>1 Do not calculate a TCP or UDP pseudo-header checksum, but instead use the value in field PHCS when determining the overall TCP or UDP checksum. |
| | 8–14 | — | Reserved |
| | 15 | PTP | Indication to the transmitter that this is a PTP packet. Enabling PTP disables per packet VLAN tag insertion. Instead, VLAN tag will be read from the DFVLAN when the PTP field is true.<br>0 Do not attempt to capture transmission event time<br>1 Valid PTP_ID field. When this packet is transmitted, capture the time of transmission.<br>Must be clear if TMR_CTRL[TE] is clear. |
| 2–3 | 0–7 | L4OS | Layer 4 header offset from start of layer 3 header. The layer 4 header starts L4OS octets after the layer 3 header if it is present. The maximum layer 3 header length supported is thus 255 bytes, which may prevent TCP/IP offload on particularly large IPv6 headers. |
| | 8–15 | L3OS | Layer 3 header offset from start of frame not including the 8 bytes for this FCB. The layer 3 header starts L3OS octets from the start of the frame including any custom preamble header that may be present. The maximum layer 2 header length supported is thus 255 bytes. |
| 4–5 | 0–15 | PHCS | Pseudo-header checksum (16-bit one's complement sum with carry wraparound, but without result inversion) for TCP or UDP packets, calculated by software. Valid only if NPH = 1. |
| 6–7 | 0–15 | VLCTL/<br>PTP_ID | VLAN control word for insertion in the transmitted VLAN tag. Valid only if VLN = 1.Tx PTP packet identification number. This number will be copied into the Tx PTP packet time stamp identification field. PTP field takes precedence over VLN field. |

## 14.6.8 Buffer Descriptors

The eTSEC buffer descriptor (BD) is modeled after the MPC8260 Fast Ethernet controller BD for ease of reuse across the PowerQUICC network processor family. Drawing from the MPC8260 FEC BD programming model, the eTSEC descriptor base registers point to the beginning of BD rings. The eTSEC BD also expands upon the MPC8260 BD model to accommodate the eTSEC's unique features. However, the 8-byte data BD format is designed to be compatible with the existing MPC8260 BD model.

The eTSEC is capable of duplicating—or extracting—data directly into the L2 cache memory. This allows the processor to quickly access critical frame information as soon as the processor is ready without having to first fetch the data from main memory, which holds the master copy. This results in substantial improvement in throughput and hence reduction in latency.

#### 14.6.8.1 Data Buffer Descriptors

Data buffers are used in the transmission and reception of Ethernet frames (see Figure 14-154). Data BDs encapsulate all information necessary for the eTSEC to transmit or receive an Ethernet frame. Within each data BD there is a status field, a data length field, and a data pointer. The BD completely describes an Ethernet packet by centralizing status information for the data packet in the status field of the BD and by containing a data BD pointer to the location of the data buffer. Software is responsible for setting up the BDs in memory. Because of pre-fetching, a minimum of four buffer descriptors per ring are required. This applies to both the transmit and the receive descriptor rings. Transmit rings are limited to a maximum size of 65536 BDs due to BD and frame data prefetching. Software also must have the data pointer pointing to a legal memory location. Within the status field, there exists an ownership bit which defines the current state of the buffer (pointed to by the data pointer). Other bits in the status field of the buffer descriptor are used to communicate status/control information between the eTSEC and the software driver.

Because there is no next BD pointer in the transmit/receive BD (see Figure 14-155), all BDs must reside sequentially in memory. The eTSEC increments the current BD location appropriately to the next BD location to be processed. There is a wrap bit in the last BD that informs the eTSEC to loop back to the beginning of the BD chain. Software must initialize the TBASE and RBASE registers that point to the beginning transmit and receive BDs for eTSEC.

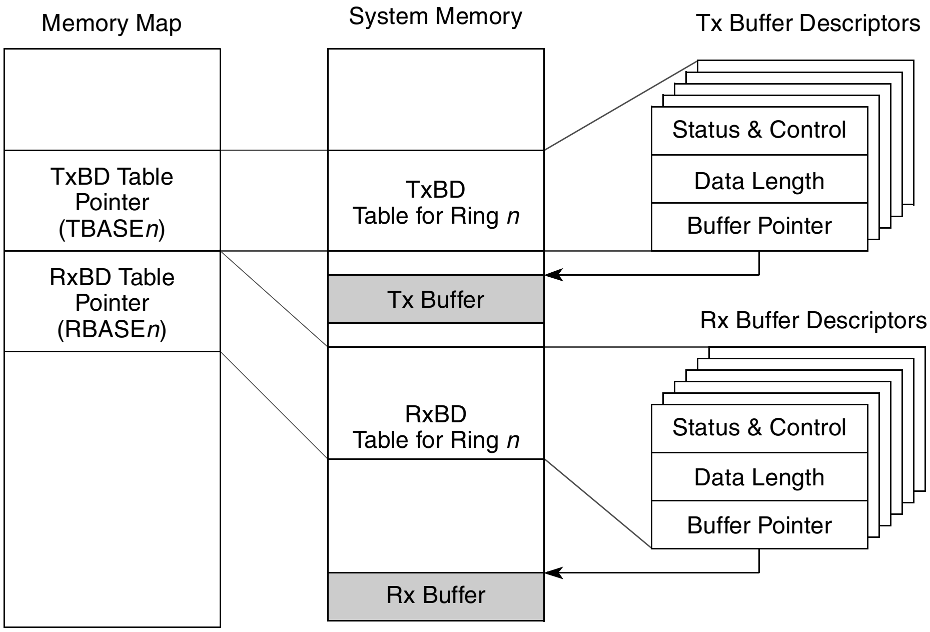


**Figure 14-154. Example of eTSEC Memory Structure for BDs**

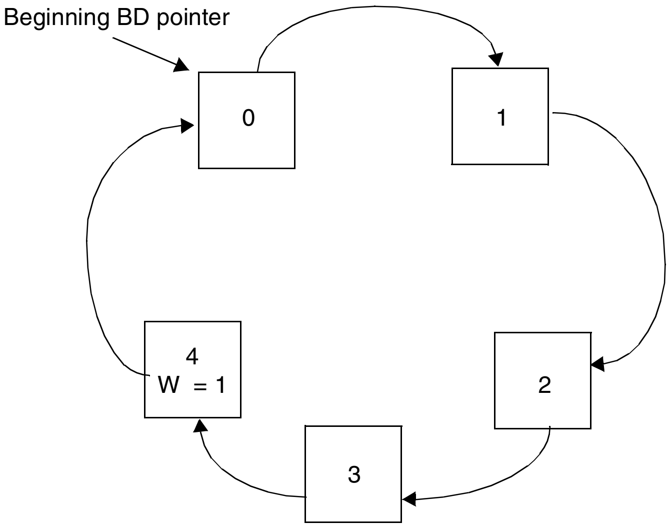


**Figure 14-155. Buffer Descriptor Ring**

### 14.6.8.2 Transmit Data Buffer Descriptors (TxBD)

Data is presented to the eTSEC for transmission by arranging it in memory buffers referenced by the TxBDs. In the TxBD the user initializes the R, PAD, W, I, L, TC, PRE, HFE, CF, and TOE bits and the length (in bytes) in the first word, and the buffer pointer in the second word. Unused fields or fields written by the eTSEC must be initialized to zero. For transmission over the FIFO interface the Ethernet specific bits (PRE, DEF, HFE, LC, CF, RL and RC) have no meaning.

The eTSEC clears the R bit in the first word of the BD after it finishes using the data buffer. The transfer status bits are then updated. Additional transmit frame status can be found in statistic counters in the MIB block.

Software must expect eTSEC to prefetch multiple TxBDs, and for TCP/IP checksumming an entire frame must be read from memory before a checksum can be computed. Accordingly, the R bit of the first TxBD in a frame must not be set until at least one entire frame can be fetched from this TxBD onwards. If eTSEC prefetches TxBDs and fails to reach a last TxBD (with bit L set), it halts further transmission from the current TxBD ring and report an underrun error as IEVENT[XFUN]; this indicates that an incomplete frame was fetched, but remained unprocessed. The relevant TBPTR register points to the next unread TxBD following the error.

Figure 14-156 defines the TxBD.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Offset + 0 | R | PAD/CRC | W | I | L | TC | PRE/DEF | 0 | HFE/LC | CF/RL | RC | | | | TOE/UN | TR |
| Offset + 2 | DATA LENGTH | | | | | | | | | | | | | | | |
| Offset + 4 | TX DATA BUFFER POINTER | | | | | | | | | | | | | | | |
| Offset + 6 | | | | | | | | | | | | | | | | |

**Figure 14-156. Transmit Buffer Descriptor**

The TxBD definition is interpreted by eTSEC hardware as if TxBDs mapped to C data structures in the manner illustrated by Figure 14-157.

```
typedef unsigned short uint_16; /* choose 16-bit native type */
typedef unsigned int uint_32; /* choose 32-bit native type */
typedef struct txbd_struct {
        uint_16 flags;
        uint_16 length;
        uint_32 bufptr;
} txbd;
```

**Figure 14-157. Mapping of TxBDs to a C Data Structure**

The TxBD fields are detailed in Table 14-169.

**Table 14-169. Transmit Data Buffer Descriptor (TxBD) Field Descriptions**

| Offset | Bits | Name | Description |
|---|---|---|---|
| 0–1 | 0 | R | Ready, written by eTSEC and user.<br>0 The data buffer associated with this BD is not ready for transmission. The user is free to manipulate this BD or its associated data buffer. The eTSEC clears this bit after the buffer is transmitted or after an error condition is encountered.<br>1 The data buffer, which is prepared for transmission by the user, was not transmitted or is currently being transmitted. No fields of this BD may be written by the user once this bit is set. |
| | 1 | PAD/CRC | Padding for frames. (Valid only while it is set in the first BD and MACCFG2[PAD enable] is cleared). If MACCFG2[PAD enable] is set, this bit is ignored.<br>0 Do not add padding to short frames.<br>1 Add PAD to frames. PAD bytes are inserted until the length of the transmitted frame equals 64 bytes. Unlike the MPC8260 which PADs up to MINFLR value, the eTSEC PADs always up to the IEEE minimum frame length of 64 bytes. CRC is always appended to frames. |
| | 2 | W | Wrap. Written by user.<br>0 The next buffer descriptor is found in the consecutive location.<br>1 The next buffer descriptor is found at the location defined in TBASE. |
| | 3 | I | Interrupt. Written by user.<br>0 No interrupt is generated after this buffer is serviced.<br>1 IEVENT[TXB] or IEVENT[TXF] are set after this buffer is serviced. These bits can cause an interrupt if they are enabled (That is, IEVENT[TXBEN] or IEVENT[TXFEN] are set). |
| | 4 | L | Last in frame. Written by user.<br>0 The buffer is not the last in the transmit frame.<br>1 The buffer is the last in the transmit frame. |

Table 14-169. Transmit Data Buffer Descriptor (TxBD) Field Descriptions (continued)

| Offset | Bits | Name | Description |
|---|---|---|---|
| 0–1 | 5 | TC | Tx CRC. Written by user. (Valid only while it is set in first BD and TxBD[PAD/CRC] is cleared and MACCFG2[PAD/CRC enable] is cleared and MACCFG2[CRC enable] is cleared.) If MACCFG2[PAD/CRC enable] is set or MACCFG2[CRC enable] is set, this bit is ignored in ethernet modes.<br>If FIFOCFG[CRCAPP] is set, this bit is ignored in FIFO modes<br>0 End transmission immediately after the last data byte with no hardware generated CRC appended, unless TxBD[PAD/CRC] is set.<br>1 Transmit the CRC sequence after the last data byte. |
| | 6 | PRE | Transmit user-defined Ethernet preamble. Written by user. Valid only if set in the first BD of a frame, and MACCFG2[PreAm TxEN] is set.<br>0 This frame does not contain Ethernet preamble bytes for transmission.<br>1 This frame includes a user-defined Ethernet preamble sequence prior to the destination address in the data buffer. |
| | | DEF | Defer indication. The eTSEC updates this bit after transmitting a frame (TxBD[L] is set)<br>0 This frame was not deferred.<br>1 This frame did not have a collision before it was sent but it was sent late because of deferring |
| | 7 | — | Reserved |
| | 8 | HFE | Huge frame enable. Written by user. Valid only if set in the first BD of a frame and MACCFG2[Huge Frame] is cleared. If MACCFG2[Huge Frame] is set, this bit is ignored.<br>0 Truncate transmit frame if its length is greater than the MAC's maximum frame length.<br>1 Allow large frames to be transmitted without truncation. |
| | | LC | Late collision. Written by the eTSEC.<br>0 No late collision.<br>1 A collision occurred after 64 bytes are sent. The eTSEC terminates the transmission and updates LC. |
| | 9 | CF | Control Frame. Written by user. Valid only if set in the first BD of a frame.<br>0 Regular frame; transmission is deferred when eTSEC is in PAUSE.<br>1 Control frame; transmission starts even if eTSEC is in PAUSE. |
| | | RL | Retransmission Limit. Written by the eTSEC.<br>0 Transmission before maximum retry limit is hit.<br>1 The transmitter failed (max. retry limit + 1) attempts to successfully send a message due to repeated collisions. The eTSEC terminates the transmission and updates RL. |
| | 10–13 | RC | Retry Count. Written by the eTSEC.<br>0 The frame is sent correctly the first time.<br>x One or more attempts where needed to send the transmit frame. If this field is 15, then 15 or more retries were needed. The Ethernet controller updates RC after sending the buffer. |

**Table 14-169. Transmit Data Buffer Descriptor (TxBD) Field Descriptions (continued)**

| Offset | Bits | Name | Description |
|---|---|---|---|
| 0–1 | 14 | UN | Underrun. Written by the eTSEC.<br>0 No underrun encountered (data was retrieved from external memory in time to send a complete frame).<br>1 The Ethernet controller encountered a transmitter underrun condition while sending the associated buffer. This could also have occurred in relation to a bus error causing IEVENT[EBERR]. The eTSEC terminates the transmission and updates UN. |
| | | TOE | TCP/IP off-load enable. Written by user. Valid only if set in the first BD of a frame.<br>0 No TCP/IP off-load acceleration is applied to the frame prior to transmission.<br>1 eTSEC looks for a TOE Frame Control Block preceding the frame, and applies TCP/IP off-load acceleration as controlled by the FCB. |
| | 15 | TR | Truncation. Written by the eTSEC. Set in the last TxBD (TxBD[L] is set) when IEVENT[BABT] occurs for a frame (a frame length greater than or equal to the value set in the maximum frame length register is encountered, the HFE bit in the BD is cleared, and MACCFG2[Huge Frame] is cleared). The frame is sent truncated. |
| 2–3 | 0–15 | Data Length | Data length is the number of octets the eTSEC should transmit from this BD's data buffer. It is never modified by the eTSEC. This field must be greater than zero, as zero indicates a BD not ready. |
| 4–7 | 0–31 | TX Data Buffer Pointer | The transmit buffer pointer contains the address of the associated data buffer. The data buffer pointer for the first BD of a TxPAL-enabled frame must be aligned on an 8-byte boundary. There are no alignment restrictions for the data buffer pointers of the second or subsequent BDs of a TxPAL-enabled frame, or for non-TxPAL frames. |

### 14.6.8.3 Receive Buffer Descriptors (RxBD)

In the RxBD the user initializes the E, I, and W bits in the first word and the pointer in second word. If the data buffer is used, the eTSEC modifies the E, L, F, M, BC, MC, LG, NO, CR, OV, and TR bits and writes the length of the used portion of the buffer in the first word. The M, BC, MC, LG, NO, CR, OV, and TR bits in the first word of the buffer descriptor are only modified by the eTSEC if the L (last BD in frame) bit is set. The first word of the RxBD contains control and status bits. Its formats are detailed below.

The number of buffer descriptors in a ring is set using the W bit to indicate that the next buffer wraps back to the beginning of the ring. See Section 14.5.3.5.5, "Maximum Frame Length Register (MAXFRM)," for information on setting the size of the buffer ring.

Figure 14-158 defines the RxBD.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Offset + 0 | E | RO1 | W | I | L | F | 0 | M | BC | MC | LG | NO | SH | CR | OV | TR |
| Offset + 2 | DATA LENGTH | | | | | | | | | | | | | | | |
| Offset + 4 | RX DATA BUFFER POINTER | | | | | | | | | | | | | | | |
| Offset + 6 | | | | | | | | | | | | | | | | |

**Figure 14-158. Receive Buffer Descriptor**

The RxBD definition is interpreted by eTSEC hardware as if RxBDs mapped to C data structures in the manner illustrated by Figure 14-159.

```
typedef unsigned short uint_16; /* choose 16-bit native type */
typedef unsigned int uint_32; /* choose 32-bit native type */
typedef struct rxbd_struct {
        uint_16 flags;
        uint_16 length;
        uint_32 bufptr;
} rxbd;
```

**Figure 14-159. Mapping of RxBDs to a C Data Structure**

Table 14-170 describes the fields of the RxBD.

**Table 14-170. Receive Buffer Descriptor Field Descriptions**

| Offset | Bits | Name | Description |
|---|---|---|---|
| 0–1 | 0 | E | Empty, written by the eTSEC (when cleared) and by the user (when set).<br>0 The data buffer associated with this BD is filled with received data, or data reception is aborted due to an error condition. The status and length fields have been updated as required.<br>1 The data buffer associated with this BD is empty, or reception is currently in progress. |
| | 1 | RO1 | Receive software ownership bit.<br>This field is reserved for use by software. This read/write bit is not modified by hardware, nor does its value affect hardware. |
| | 2 | W | Wrap, written by user.<br>0 The next buffer descriptor is found in the consecutive location.<br>1 The next buffer descriptor is found at the location defined in RBASE. |
| | 3 | I | Interrupt, written by user.<br>0 No interrupt is generated after this buffer is serviced.<br>1 IEVENT[RXB] or IEVENT[RXF] are set after this buffer is serviced. This bit can cause an interrupt if enabled (IMASK[RXBEN] or IMASK[RXFEN]). If the user wants to be interrupted only if RXF occurs, then the user must disable RXB (IMASK[RXBEN] is cleared) and enable RXF (IMASK[RXFEN] is set). |
| | 4 | L | Last in frame, written by the eTSEC.<br>0 The buffer is not the last in a frame.<br>1 The buffer is the last in a frame. |
| | 5 | F | First in frame, written by the eTSEC.<br>0 The buffer is not the first in a frame.<br>1 The buffer is the first in a frame. |
| | 6 | — | Reserved |
| | 7 | M | Miss, written by the eTSEC. (This bit is valid only if the L-bit is set and eTSEC is in promiscuous mode.) This bit is set by the eTSEC for frames that were accepted in promiscuous mode, but were flagged as a “miss” by the internal address recognition; thus, while in promiscuous mode, the user can use the M-bit to quickly determine whether the frame was destined to this station.<br>0 The frame was received because of an address recognition hit.<br>1 The frame was received because of promiscuous mode. |

**Table 14-170. Receive Buffer Descriptor Field Descriptions (continued)**

| Offset | Bits | Name | Description |
|---|---|---|---|
| 0–1 | 8 | BC | Broadcast. Written by the eTSEC. (Only valid if L is set.) Is set if the DA is broadcast (FF-FF-FF-FF-FF-FF). |
| | 9 | MC | Multicast. Written by the eTSEC. (Only valid if L is set.) Is set if the DA is multicast and not BC. |
| | 10 | LG | Rx frame length violation, written by the eTSEC (only valid if L is set).<br>A frame length greater than or equal to the maximum frame length was recognized; in this case LG is set regardless of the setting of MACCFG2[Huge Frame]. If MACCFG2[Huge Frame] is cleared, the frame is truncated to the value programmed in the maximum frame length register. This bit is valid only if the L bit is set. |
| | 11 | NO | Rx non-octet aligned frame, written by the eTSEC (only valid if L is set).<br>A frame that contained a number of bits not divisible by eight was received. |
| | 12 | SH | Short frame, written by the eTSEC (only valid if L is set). A frame length less than the minimum 64B that is defined for ethernet. was recognized, provided RCTRL[RSF] is set. |
| | 13 | CR | Rx CRC error, written by the eTSEC (only valid if L is set).<br>This frame contains a CRC error and is an integral number of octets in length.This bit is also set if a receive code group error is detected. |
| | 14 | OV | Overrun, written by the eTSEC (only valid if L is set).<br>A receive FIFO overrun occurred during frame reception. If this bit is set, the other status bits, M, LG, NO, CR and TR lose their normal meaning and are zero. |
| | 15 | TR | Truncation, written by the eTSEC (only valid if L is set).<br>Set if the receive frame is truncated. This can happen if a frame length greater than the maximum frame length is received and MACCFG2[Huge Frame] is cleared. If this bit is set, the frame must be discarded and the other error bits must be ignored as they may be incorrect.<br>This bit is not set when in FIFO mode as truncation cannot occur. |
| 2–3 | 0–15 | Data Length | Data length, written by the eTSEC.<br>Data length is the number of octets written by the eTSEC into this BD’s data buffer if L is cleared (the value is equal to MRBLR), or, if L is set, the length of the frame including CRC, FCB (if RCTRL[PRSDEP > 00), preamble (if MACCFG2[PreAmRxEn]=1), timestamp (if RCTRL[TS]=1) and any padding (RCTRL[PAL]). |
| 4–7 | 0–31 | RX Data Buffer Pointer | Receive buffer pointer, written by the user.<br>The receive buffer pointer, which always points to the first location of the associated data buffer, must be 8-byte aligned. For best performance, use 64-byte aligned receive buffer pointer addresses. The buffer must reside in memory external to the eTSEC. |

## 14.7 Initialization/Application Information

### 14.7.1 Interface Mode Configuration

This section describes how to configure the eTSEC in different supported interface modes. These include the following:

- MII
- RMII

- GMII
- RGMII
- SGMII
- TBI
- RTBI
- 8-bit FIFO

The pinout, the data registers that must be initialized, as well as speed selection options are described.

#### 14.7.1.1 MII Interface Mode

Table 14-171 describes the signal configurations required for MII interface mode.

Table 14-171. MII Interface Mode Signal Configuration

| eTSEC Signals | | | MII Interface | | |
|---|---|---|---|---|---|
| | | | Frequency [MHz] 25 | | |
| | | | Voltage [V] 3.3 | | |
| **Signals** | **I/O** | **No. of Signals** | **Signals** | **I/O** | **No. of Signals** |
| GTX_CLK | O | 1 | leave unconnected | | |
| TX_CLK | I | 1 | TX_CLK | I | 1 |
| TxD[0] | O | 1 | TxD[0] | O | 1 |
| TxD[1] | O | 1 | TxD[1] | O | 1 |
| TxD[2] | O | 1 | TxD[2] | O | 1 |
| TxD[3] | O | 1 | TxD[3] | O | 1 |
| TxD[4] | O | 1 | leave unconnected | | |
| TxD[5] | O | 1 | leave unconnected | | |
| TxD[6] | O | 1 | leave unconnected | | |
| TxD[7] | O | 1 | leave unconnected | | |
| TX_EN | O | 1 | TX_EN | O | 1 |
| TX_ER | O | 1 | TX_ER | O | 1 |
| RX_CLK | I | 1 | RX_CLK | I | 1 |
| RxD[0] | I | 1 | RxD[0] | I | 1 |
| RxD[1] | I | 1 | RxD[1] | I | 1 |
| RxD[2] | I | 1 | RxD[2] | I | 1 |
| RxD[3] | I | 1 | RxD[3] | I | 1 |
| RxD[4] | I | 1 | not used | | |
| RxD[5] | I | 1 | not used | | |
| RxD[6] | I | 1 | not used | | |

Table 14-171. MII Interface Mode Signal Configuration (continued)

| eTSEC Signals | | | MII Interface | | |
|---|---|---|---|---|---|
| | | | Frequency [MHz] 25 | | |
| | | | Voltage [V] 3.3 | | |
| **Signals** | **I/O** | **No. of Signals** | **Signals** | **I/O** | **No. of Signals** |
| RxD[7] | I | 1 | not used | | |
| RX_DV | I | 1 | RX_DV | I | 1 |
| RX_ER | I | 1 | RX_ER | I | 1 |
| COL | I | 1 | COL | I | 1 |
| CRS | I | 1 | CRS | I | 1 |
| **Sum** | | 25 | **Sum** | | 16 |

Table 14-172 describes the shared signals of the MII interface.

Table 14-172. Shared MII Signals

| eTSEC Signals | I/O | No. of Signals | MII Signals | I/O | No. of Signals | Function |
|---|---|---|---|---|---|---|
| MDIO | I/O | 1 | MDIO | I/O | 1 | Management interface I/O |
| MDC | O | 1 | MDC | O | 1 | Management interface clock |
| ECGTX_CLK125 | I | 1 | not used | I | 0 | Reference clock |
| **Sum** | | 3 | **Sum** | | 2 | |

Table 14-173 describes the register initializations required to configure the eTSEC in MII mode.

Table 14-173. MII Mode Register Initialization Steps

| |
|---|
| Set Soft_Reset,<br>MACCFG1[1000_0000_0000_0000_0000_0000_0000_0000] |
| Clear Soft_Reset,<br>MACCFG1[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize MACCFG2, for MII, half duplex operation.<br>Set I/F Mode bit,<br>MACCFG2[0000_0000_0000_0000_0111_0001_0000_0100]<br>(This example has Full Duplex = 0, Preamble count = 7, PAD/CRC append = 1) |
| Initialize ECNTRL,<br>ECNTRL[0000_0000_0000_0000_0001_0000_0000_0000]<br>(This example has Statistics Enable = 1) |
| Initialize MAC Station Address,<br>MACSTNADDR2[0110_0000_0000_0010_0000_0000_0000_0000]<br>Set station address to 02_60_8C_87_65_43, for example. |
| Initialize MAC Station Address,<br>MACSTNADDR1[0100_0011_0110_0101_1000_0111_1000_1100]<br>Set station address to 02_60_8C_87_65_43, for example. |

**Table 14-173. MII Mode Register Initialization Steps (continued)**

| |
|---|
| Assign a Physical address to the TBI so as to not conflict with the external PHY Physical address,<br>TBIPA[0000_0000_0000_0000_0000_0000_0000_0101]<br>Set to 05, for example. |
| Reset the management interface.<br>MIIMCFG[1000_0000_0000_0000_0000_0000_0000_0111] |
| Setup the MII Mgmt clock speed,<br>MIIMCFG[0000_0000_0000_0000_0000_0000_0000_0101]<br>set source clock divide by 14 for example to insure that MDC clock speed is not greater than 2.5 MHz |
| Read MII Mgmt Indicator register and check for Busy = 0,<br>MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000]<br>This indicates that the eTSEC MII Mgmt bus is idle. |
| Set up the MII Mgmt for a write cycle to the external PHY Auxiliary Control and Status Register to configure the PHY through the Management interface (overrides configuration signals of the PHY).<br>MIIMADD[0000_0000_0000_0000_0000_0000_0001_1100] |
| Perform an MII Mgmt write cycle to the external PHY<br>Writing to MII Mgmt Control with 16-bit data intended for the external PHY register,<br>MIIMCON[0000_0000_0000_0000_0000_0000_0000_0100] |
| Check to see if MII Mgmt write is complete<br>Read MII Mgmt Indicator register and check for Busy = 0,<br>MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000]<br>This indicates that the write cycle was completed. |
| Set up the MII Mgmt for a write cycle to the external PHY Extended PHY control register #1 to set up the interface mode selection.<br>MIIMADD[0000_0000_0000_0000_0000_0000_0001_0111] |
| Perform an MII Mgmt write cycle to the external PHY.<br>Write to MII Mgmt Control with 16-bit data intended for the external PHY register,<br>MIIMCON[0000_0000_0000_0000_0000_0000_0000_0000] |
| Check to see if MII Mgmt write is complete.<br>Read MII Mgmt Indicator register and check for Busy = 0,<br>MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000]<br>This indicates that the write cycle was completed. |
| Set up the MII Mgmt for a write cycle to the external PHY Mode control register to set up the interface mode selection.<br>MIIMADD[0000_0000_0000_0000_0000_0000_0000_0000] |
| Perform an MII Mgmt write cycle to the external PHY.<br>Write to MII Mgmt Control with 16-bit data intended for the external PHY register,<br>MIIMCON[0000_0000_0000_0000_00uu_00uu_0u00_0000]<br>where u is user defined based on desired configuration. |
| Check to see if MII Mgmt write is complete<br>Read MII Mgmt Indicator register and check for Busy = 0,<br>MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000]<br>This indicates that the write cycle was completed. |
| If auto-negotiation was enabled in the PHY, check to see if PHY has completed Auto-Negotiation.<br>Set up the MII Mgmt for a read cycle to PHY MII Mgmt register (write the PHY address and Register address),<br>MIIMADD[0000_0000_0000_0000_0000_0000_0000_0001]<br>The PHY Status register is at address 0x1 and in this case the PHY Address is 0x00. |

### Table 14-173. MII Mode Register Initialization Steps (continued)

| |
|---|
| Perform an MII Mgmt read cycle of Status Register.<br>Clear MIIMCOM[Read Cycle].<br>Set MIIMCOM[Read Cycle].<br>(Uses the PHY address (0) and Register address (1) placed in MIIMADD register),<br>When MIIMIND[BUSY]=0,<br>read the MIIMSTAT register and check bit 10 (AN Done and Link is up)<br>MIIMSTAT ---> [0000_0000_0000_0000_0000_0000_0010_0100]<br>Other information about the link is also returned.(Extend Status, No pre, Remote Fault, An Ability, Link status, extend Ability) |
| Check auto-negotiation attributes in the PHY as necessary. |
| Clear IEVENT register,<br>IEVENT[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize IMASK (Optional)<br>IMASK[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize MACnADDR1/2 (Optional)<br>MACnADDR1/2[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize GADDR*n* (Optional)<br>GADDR*n*[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize RCTRL (Optional)<br>RCTRL[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize DMACTRL (Optional)<br>DMACTRL[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize (Empty) Transmit Descriptor ring and fill buffers with Data<br>Initialize TBASE0–TBASE7,<br>TBASE0–TBASE7[LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_L000] |
| Initialize (Empty) Receive Descriptor ring and fill with empty buffers<br>Initialize RBASE0–RBASE7,<br>RBASE0–RBASE7[LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_L000] |
| Enable Transmit Queues<br>Initialize TQUEUE |
| Enable Receive Queues<br>Initialize RQUEUE |
| Enable Rx and Tx,<br>MACCFG1[0000_0000_0000_0000_0000_0000_0000_0101] |

### 14.7.1.2 GMII Interface Mode

Table 14-174 describes the signal configurations required for GMII interface mode.

Table 14-174. GMII Interface Mode Signal Configuration

| eTSEC Signal s | | | GMII Interface | | |
|---|---|---|---|---|---|
| | | | Frequency [MHz] 125 | | |
| | | | Voltage [V] 3.3 | | |
| **Signals** | **I/O** | **No. of Signals** | **Signals** | **I/O** | **No. of Signals** |
| GTX_CLK | O | 1 | GTX_CLK | O | 1 |
| TX_CLK | I | 1 | TX_CLK | I | 1 |
| TxD[0] | O | 1 | TxD[0] | O | 1 |
| TxD[1] | O | 1 | TxD[1] | O | 1 |
| TxD[2] | O | 1 | TxD[2] | O | 1 |
| TxD[3] | O | 1 | TxD[3] | O | 1 |
| TxD[4] | O | 1 | TxD[4] | O | 1 |
| TxD[5] | O | 1 | TxD[5] | O | 1 |
| TxD[6] | O | 1 | TxD[6] | O | 1 |
| TxD[7] | O | 1 | TxD[7] | O | 1 |
| TX_EN | O | 1 | TX_EN | O | 1 |
| TX_ER | O | 1 | TX_ER | O | 1 |
| RX_CLK | I | 1 | RX_CLK | I | 1 |
| RxD[0] | I | 1 | RxD[0] | I | 1 |
| RxD[1] | I | 1 | RxD[1] | I | 1 |
| RxD[2] | I | 1 | RxD[2] | I | 1 |
| RxD[3] | I | 1 | RxD[3] | I | 1 |
| RxD[4] | I | 1 | RxD[4] | I | 1 |
| RxD[5] | I | 1 | RxD[5] | I | 1 |
| RxD[6] | I | 1 | RxD[6] | I | 1 |
| RxD[7] | I | 1 | RxD[7] | I | 1 |
| RX_DV | I | 1 | RX_DV | I | 1 |
| RX_ER | I | 1 | RX_ER | I | 1 |
| COL | I | 1 | not used | | |
| CRS | I | 1 | not used | | |
| **Sum** | | 25 | **Sum** | | 23 |

Table 14-175 describes the shared signals of the GMII interface.

Table 14-175. Shared GMII Signals

| eTSEC Signals | I/O | No. of Signals | GMII Signals | I/O | No. of Signals | Function |
|---|---|---|---|---|---|---|
| MDIO | I/O | 1 | MDIO | I/O | 1 | Management interface I/O |
| MDC | O | 1 | MDC | O | 1 | Management interface clock |
| ECGTX_CLK125 | I | 1 | GTX_CLK125 | I | 1 | Reference clock |
| **Sum** | | 3 | **Sum** | | 3 | |

Table 14-176 describes the register initializations required to configure the eTSEC in GMII mode.

Table 14-176. GMII Mode Register Initialization Steps

| |
|---|
| Set Soft_Reset,<br>MACCFG1[1000_0000_0000_0000_0000_0000_0000_0000] |
| Clear Soft_Reset,<br>MACCFG1[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize MACCFG2, for GMII, Full duplex operation.<br>Set I/F Mode bit.<br>MACCFG2[0000_0000_0000_0000_0111_0010_0000_0101]<br>(This example has Full Duplex = 1, Preamble count = 7, PAD/CRC append = 1) |
| Initialize ECNTRL,<br>ECNTRL[0000_0000_0000_0000_0001_0000_0000_0000]<br>(This example has Statistics Enable = 1) |
| Initialize MAC Station Address,<br>MACSTNADDR2[0110_0000_0000_0010_0000_0000_0000_0000]<br>Set station address to 02_60_8C_87_65_43, for example. |
| Initialize MAC Station Address,<br>MACSTNADDR1[0100_0011_0110_0101_1000_0111_1000_1100]<br>Set station address to 02_60_8C_87_65_43, for example. |
| Assign a Physical address to the TBI so as to not conflict with the external PHY Physical address,<br>TBIPA[0000_0000_0000_0000_0000_0000_0000_0101]<br>Set to 05, for example. |
| Reset the management interface,<br>MIIMCFG[1000_0000_0000_0000_0000_0000_0000_0111] |
| Setup the MII Mgmt clock speed,<br>MIIMCFG[0000_0000_0000_0000_0000_0000_0000_0101]<br>set source clock divide by 14 for example to insure that MDC clock speed is not greater than 2.5 MHz |
| Read MII Mgmt Indicator register and check for Busy = 0,<br>MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000]<br>This indicates that the eTSEC MII Mgmt bus is idle. |
| Set up the MII Mgmt for a write cycle to the external PHY Auxiliary Control and Status Register to configure the PHY through the Management interface (overrides configuration signals of the PHY),<br>MIIMADD[0000_0000_0000_0000_0000_0000_0001_1100] |
| Perform an MII Mgmt write cycle to the external PHY.<br>Write to MII Mgmt Control with 16-bit data intended for the external PHY register,<br>MIIMCON[0000_0000_0000_0000_0000_0000_0000_0100] |

**Table 14-176. GMII Mode Register Initialization Steps (continued)**

| |
|---|
| Check to see if MII Mgmt write is complete.<br>Read MII Mgmt Indicator register and check for Busy = 0,<br>MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000]<br>This indicates that the write cycle was completed |
| Set up the MII Mgmt for a write cycle to the external PHY Extended PHY control register #1 to set up the interface mode selection<br>MIIMADD[0000_0000_0000_0000_0000_0000_0001_0111] |
| Perform an MII Mgmt write cycle to the external PHY.<br>Write to MII Mgmt Control with 16-bit data intended for the external PHY register,<br>MIIMCON[0000_0000_0000_0000_0000_0000_0000_0000] |
| Check to see if MII Mgmt write is complete.<br>Read MII Mgmt Indicator register and check for Busy = 0,<br>MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000]<br>This indicates that the write cycle was completed. |
| Set up the MII Mgmt for a write cycle to the external PHY Mode control register to set up the interface mode selection,<br>MIIMADD[0000_0000_0000_0000_0000_0000_0000_0000] |
| Perform an MII Mgmt write cycle to the external PHY.<br>Write to MII Mgmt Control with 16-bit data intended for the external PHY register,<br>MIIMCON[0000_0000_0000_0000_000u_00u1_0100_0000]<br>where u is user defined based on desired configuration. |
| Check to see if MII Mgmt write is complete.<br>Read MII Mgmt Indicator register and check for Busy = 0,<br>MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000]<br>This indicates that the write cycle was completed. |
| If auto-negotiation was enabled in the PHY, check to see if PHY has completed Auto-Negotiation.<br>Set up the MII Mgmt for a read cycle to PHY MII Mgmt register (write the PHY address and Register address),<br>MIIMADD[0000_0000_0000_0000_0000_0000_0000_0001]<br>The PHY Status register is at address 0x1 and in this case the PHY Address is 0x00 |
| Perform an MII Mgmt read cycle of Status Register.<br>Clear MIIMCOM[Read Cycle].<br>Set MIIMCOM[Read Cycle].<br>(Uses the PHY address (0) and Register address (1) placed in MIIMADD register),<br>When MIIMIND[BUSY]=0,<br>Read the MIIMSTAT register and check bit 10 (AN Done and Link is up),<br>MIIMSTAT ---> [0000_0000_0000_0000_0000_0000_0010_0100]<br>Other information about the link is also returned.(Extend Status, No pre, Remote Fault, An Ability, Link status, extend Ability) |
| Check auto-negotiation attributes in the PHY as necessary. |
| Clear IEVENT register,<br>IEVENT[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize IMASK (Optional)<br>IMASK[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize MACnADDR1/2 (Optional)<br>MACnADDR1/2[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize GADDR*n* (Optional)<br>GADDR*n*[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize RCTRL (Optional)<br>RCTRL[0000_0000_0000_0000_0000_0000_0000_0000] |

**Table 14-176. GMII Mode Register Initialization Steps (continued)**

| |
|---|
| Initialize DMACTRL (Optional)<br>DMACTRL[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize (Empty) Transmit Descriptor ring and fill buffers with Data<br>Initialize TBASE0–TBASE7,<br>TBASE0–TBASE7[LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_L000] |
| Initialize (Empty) Receive Descriptor ring and fill with empty buffers<br>Initialize RBASE0–RBASE7,<br>RBASE0–RBASE7[LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_L000] |
| Enable Transmit Queues<br>Initialize TQUEUE |
| Enable Receive Queues<br>Initialize RQUEUE |
| Enable Rx and Tx,<br>MACCFG1[0000_0000_0000_0000_0000_0000_0000_0101] |

### 14.7.1.3 TBI Interface Mode

Table 14-177 describes the signal configurations required for TBI interface mode.

Table 14-177. TBI Interface Mode Signal Configuration

| eTSEC Signals | | | TBI Interface | | |
|---|---|---|---|---|---|
| | | | Frequency [MHz] 62.5 | | |
| | | | Voltage [V] 3.3 | | |
| **Signals** | **I/O** | **No. of Signals** | **Signals** | **I/O** | **No. of Signals** |
| GTX_CLK | O | 1 | GTX_CLK | O | 1 |
| TX_CLK | I | 1 | RX_CLK1 | I | 1 |
| TxD[0] | O | 1 | TCG[0] | O | 1 |
| TxD[1] | O | 1 | TCG[1] | O | 1 |
| TxD[2] | O | 1 | TCG[2] | O | 1 |
| TxD[3] | O | 1 | TCG[3] | O | 1 |
| TxD[4] | O | 1 | TCG[4] | O | 1 |
| TxD[5] | O | 1 | TCG[5] | O | 1 |
| TxD[6] | O | 1 | TCG[6] | O | 1 |
| TxD[7] | O | 1 | TCG[7] | O | 1 |
| TX_EN | O | 1 | TCG[8] | O | 1 |
| TX_ER | O | 1 | TCG[9] | O | 1 |
| RX_CLK | I | 1 | RX_CLK0 | I | 1 |
| RxD[0] | I | 1 | RCG[0] | I | 1 |
| RxD[1] | I | 1 | RCG[1] | I | 1 |
| RxD[2] | I | 1 | RCG[2] | I | 1 |
| RxD[3] | I | 1 | RCG[3] | I | 1 |
| RxD[4] | I | 1 | RCG[4] | I | 1 |
| RxD[5] | I | 1 | RCG[5] | I | 1 |
| RxD[6] | I | 1 | RCG[6] | I | 1 |
| RxD[7] | I | 1 | RCG[7] | I | 1 |
| RX_DV | I | 1 | RCG[8] | I | 1 |
| RX_ER | I | 1 | RCG[9] | I | 1 |
| COL | I | 1 | not used | | |
| CRS | I | 1 | SDET | I | 1 |
| **Sum** | | 25 | **Sum** | | 24 |

Table 14-178 describes the shared signals for the TBI interface.

**Table 14-178. Shared TBI Signals**

| eTSEC Signals | I/O | No. of Signals | GMII Signals | I/O | No. of Signals | Function |
|---|---|---|---|---|---|---|
| MDIO | I/O | 1 | MDIO | I/O | 1 | Management interface I/O |
| MDC | O | 1 | MDC | O | 1 | Management interface clock |
| ECGTX_CLK125 | I | 1 | GTX_CLK125 | I | 1 | Reference clock |
| **Sum** |  | 3 | **Sum** |  | 3 |  |

Table 14-179 describes the register initializations required to configure the eTSEC in TBI mode.

**Table 14-179. TBI Mode Register Initialization Steps**

| |
|---|
| Set Soft_Reset,<br>MACCFG1[1000_0000_0000_0000_0000_0000_0000_0000] |
| Clear Soft_Reset,<br>MACCFG1[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize MACCFG2,<br>MACCFG2[0000_0000_0000_0000_0111_0010_0000_0101]<br>(I/F Mode = 2, Full Duplex = 1) |
| Initialize ECNTRL,<br>ECNTRL[0000_0000_0000_0000_0001_0000_0000_0000]<br>(This example has Statistics Enable = 1) |
| Initialize MAC Station Address<br>MACSTNADDR2[0110_0000_0000_0010_0000_0000_0000_0000]<br>to 02608C:876543, for example. |
| Initialize MAC Station Address<br>MACSTNADDR1[0100_0011_0110_0101_1000_0111_1000_1100]<br>to 02608C:876543, for example. |
| Assign a Physical address to the TBI,<br>TBIPA[0000_0000_0000_0000_0000_0000_0001_0000]<br>set to 16, for example. |
| Setup the MII Mgmt clock speed,<br>MIIMCFG[0000_0000_0000_0000_0000_0000_0000_0101]<br>set source clock divide by 14 for example to insure that MDC clock speed is not greater than 2.5 MHz |
| Read MII Mgmt Indicator register and check for Busy = 0,<br>MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000]<br>This indicates that the eTSEC MII Mgmt bus is idle. |
| Set up the MII Mgmt for a read cycle to TBI’s Control register (write the TBI address and Register address),<br>MIIMADD[0000_0000_0000_0000_0001_0000_0000_0000]<br>The control register (CR) is at offset address 0x0 from TBIPA. |

**Table 14-179. TBI Mode Register Initialization Steps (continued)**

| |
|---|
| Perform an MII Mgmt read cycle to verify state of TBI Control Register(0ptional)<br>Clear MIIMCOM[Read Cycle]<br>Set MIIMCOM[Read Cycle]<br>(Uses the TBI address and Register address placed in MIIMADD register),<br>When MIIMIND[BUSY]=0,<br>read the MIISTAT and look for AN Enable and other bit information. |
| Set up the MII Mgmt for a write cycle to TBI's AN Advertisement register (write the PHY address and Register address),<br>MIIMADD[0000_0000_0000_0000_0001_0000_0000_0100]<br>The AN Advertisement register is at offset address 0x04 from the TBI's address. (in this case 0x10) |
| Perform an MII Mgmt write cycle to TBI.<br>Writing to MII Mgmt Control with 16-bit data intended for TBI's AN Advertisement register,<br>MIIMCON[0000_0000_0000_0000_0000_0001_1010_0000]<br>This advertises to the Link Partner that the TBI supports PAUSE and Full Duplex mode and does not support Half Duplex mode. |
| Check to see if MII Mgmt write is complete.<br>Read MII Mgmt Indicator register and check for Busy = 0,<br>MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000]<br>This indicates that the write cycle was completed. |
| Set up the MII Mgmt for a write cycle to TBI's Control register (write the PHY address and Register address),<br>MIIMADD[0000_0000_0000_0000_0001_0000_0000_0000]<br>the control register (CR) is at offset address 0x00 from the TBI's address. (in this case 0x10) |
| Perform an MII Mgmt write cycle to TBI.<br>Writing to MII Mgmt Control with 16-bit data intended for TBI's Control register,<br>MIIMCON[0000_0000_0000_0000_0001_0010_0000_0000]<br>This enables the TBI to restart Auto-Negotiations using the configuration set in the AN Advertisement register. |
| Check to see if MII Mgmt write is complete.<br>Read MII Mgmt Indicator register and check for Busy = 0,<br>MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000]<br>This indicates that the write cycle was completed. |
| Check to see if PHY has completed Auto-Negotiation.<br>Set up the MII Mgmt for a read cycle to PHY MII Mgmt register (write the PHY address and Register address),<br>MIIMADD[0000_0000_0000_0000_0001_0000_0000_0001]<br>The PHY Status control register is at address 0x1 and in this case the PHY Address is 0x10. |
| Perform an MII Mgmt read cycle of Status Register.<br>Clear MIIMCOM[Read Cycle]<br>Set MIIMCOM[Read Cycle]<br>(Uses the PHY address (2) and Register address (2) placed in MIIMADD register),<br>When MIIMIND[BUSY]=0,<br>read the MIIMSTAT register and check bit 10 (AN Done)<br>MIIMSTAT ---> [0000_0000_0000_0000_0000_0000_0010_0000]<br>Other information about the link is also returned. (Extend Status, No pre, Remote Fault, An Ability, Link status, extend Ability) |

Table 14-179. TBI Mode Register Initialization Steps (continued)

| |
|---|
| Perform an MII Mgmt read cycle of AN Expansion Register.<br>Setup MIIMADD[0000_0000_0000_0000_0001_0000_0000_0110]<br>Clear MIIMCOM[Read Cycle]<br>Set MIIMCOM[Read Cycle]<br>(Uses the PHY address (0x10) and Register address (6) placed in MIIMADD register),<br>When MIIMIND[BUSY]=0,<br>read the MII Mgmt AN Expansion register and check bits 13 and 14 (NP Able and Page Rx'd)<br>MII Mgmt AN Expansion ---> [0000_0000_0000_0000_0000_0000_0000_0110] |
| Perform an MII Mgmt read cycle of AN Link Partner Base Page Ability Register. (Optional)<br>Setup MIIMADD[0000_0000_0000_0000_0001_0000_0000_0101]<br>Clear MIIMCOM[Read Cycle]<br>Set MIIMCOM[Read Cycle]<br>(Uses the PHY address (0x10) and Register address (5) placed in MIIMADD register),<br>When MIIMIND[BUSY]=0,<br>read the MII Mgmt AN Link Partner Base Page Ability register and check bits 9 and 10. (Half and Full Duplex)<br>MII Mgmt AN Link Partner Base Page Ability ---> [0000_0000_0000_0000_0000_000x_x110_0000] |
| Clear IEVENT register,<br>IEVENT[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize IMASK (Optional)<br>IMASK[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize MACnADDR1/2 (Optional)<br>MACnADDR1/2[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize GADDR*n* (Optional)<br>GADDR*n*[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize RCTRL (Optional)<br>RCTRL[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize DMACTRL (Optional)<br>DMACTRL[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize (Empty) Transmit Descriptor ring and fill buffers with Data<br>Initialize TBASE0–TBASE7,<br>TBASE0–TBASE7[LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_L000] |
| Initialize (Empty) Receive Descriptor ring and fill with empty buffers<br>Initialize RBASE0–RBASE7,<br>RBASE0–RBASE7[LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_L000] |
| Enable Transmit Queues<br>Initialize TQUEUE |
| Enable Receive Queues<br>Initialize RQUEUE |
| Enable Rx and Tx,<br>MACCFG1[0000_0000_0000_0000_0000_0000_0000_0101] |

### 14.7.1.4 RGMII Interface Mode

Table 14-180 shows the signals configurations required for RGMII interface mode.

**Table 14-180. RGMII Interface Mode Signal Configuration**

| eTSEC Signals | | | RGMII Interface | | |
|---|---|---|---|---|---|
| | | | Frequency [MHz] 125 | | |
| | | | Voltage [V] 2.5 | | |
| **Signals** | **I/O** | **No. of Signals** | **Signals** | **I/O** | **No. of Signals** |
| GTX_CLK | O | 1 | GTX_CLK | O | 1 |
| TX_CLK | I | 1 | not used | | |
| TxD[0] | O | 1 | TxD[0]/TxD[4] | O | 1 |
| TxD[1] | O | 1 | TxD[1]/TxD[5] | O | 1 |
| TxD[2] | O | 1 | TxD[2]/TxD[6] | O | 1 |
| TxD[3] | O | 1 | TxD[3]/TxD[7] | O | 1 |
| TxD[4] | O | 1 | leave unconnected | | |
| TxD[5] | O | 1 | leave unconnected | | |
| TxD[6] | O | 1 | leave unconnected | | |
| TxD[7] | O | 1 | leave unconnected | | |
| TX_EN | O | 1 | TX_CTL (TX_EN/TX_ERR) | O | 1 |
| TX_ER | O | 1 | leave unconnected | | |
| RX_CLK | I | 1 | RX_CLK | I | 1 |
| RxD[0] | I | 1 | RxD[0]/RxD[4] | I | 1 |
| RxD[1] | I | 1 | RxD[1]/RxD[5] | I | 1 |
| RxD[2] | I | 1 | RxD[2]/RxD[6] | I | 1 |
| RxD[3] | I | 1 | RxD[3]/RxD[7] | I | 1 |
| RxD[4] | I | 1 | not used | | |
| RxD[5] | I | 1 | not used | | |
| RxD[6] | I | 1 | not used | | |
| RxD[7] | I | 1 | not used | | |
| RX_DV | I | 1 | RX_CTL (RX_DV/RX_ERR) | I | 1 |
| RX_ER | I | 1 | not used | | |
| COL | I | 1 | not used | | |
| CRS | I | 1 | not used | | |
| **Sum** | | 25 | **Sum** | | 12 |

Table 14-181 describes the shared signals for the RGMII interface.

**Table 14-181. Shared RGMII Signals**

| eTSEC Signals | I/O | No. of Signals | GMII Signals | I/O | No. of Signals | Function |
|---|---|---|---|---|---|---|
| MDIO | I/O | 1 | MDIO | I/O | 1 | Management interface I/O |
| MDC | O | 1 | MDC | O | 1 | Management interface clock |
| GTX_CLK125 | I | 1 | GTX_CLK125 | I | 1 | Reference clock |
| **Sum** | | 3 | **Sum** | | 3 | |

Table 14-182 describes the register initializations required to configure the eTSEC in RGMII mode.

**Table 14-182. RGMII Mode Register Initialization Steps**

| |
|---|
| Set Soft_Reset,<br>MACCFG1[1000_0000_0000_0000_0000_0000_0000_0000] |
| Clear Soft_Reset,<br>MACCFG1[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize MACCFG2,<br>MACCFG2[0000_0000_0000_0000_0111_0010_0000_0101]<br>(I/F Mode = 2, Full Duplex = 1) |
| Initialize ECNTRL,<br>ECNTRL[0000_0000_0000_0000_0001_0000_0000_0000]<br>(This example has RGMII 10Mbps mode, Statistics Enable = 1) |
| Initialize MAC Station Address,<br>MACSTNADDR2[0110_0000_0000_0010_0000_0000_0000_0000]<br>to 02608C:876543, for example. |
| Initialize MAC Station Address,<br>MACSTNADDR1[0100_0011_0110_0101_1000_0111_1000_1100]<br>to 02608C:876543, for example. |
| Assign a Physical address to the TBI,<br>TBIPA[0000_0000_0000_0000_0000_0000_0001_0000]<br>set to 16, for example. |
| Setup the MII Mgmt clock speed,<br>MIIMCFG[0000_0000_0000_0000_0000_0000_0000_0101]<br>Set source clock divide by 14, for example, to insure that TSEC_MDC clock speed is not greater than 2.5 MHz. |
| Read MII Mgmt Indicator register and check for Busy = 0,<br>MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000]<br>This indicates that the eTSEC MII Mgmt bus is idle. |
| Set up the MII Mgmt for a write cycle to external the PHY AN Advertisement register (write the PHY address and Register address),<br>MIIMADD[0000_0000_0000_0000_0001_0001_0000_0100]<br>The AN Advertisement register is at offset address 0x04 from the external PHY address. (in this case 0x11) |

**Table 14-182. RGMII Mode Register Initialization Steps (continued)**

| |
|---|
| Perform an MII Mgmt write cycle to the external PHY.<br>Write to MII Mgmt Control with 16-bit data intended for the external PHY AN Advertisement register,<br>MIIMCON[0000_0000_0000_0000_u0uu_uuuu_uuuu_uuuu]<br>Where u must be selected by the user for proper system configuration. |
| Check to see if MII Mgmt write is complete.<br>Read MII Mgmt Indicator register and check for Busy = 0,<br>MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000]<br>This indicates that the write cycle was completed. |
| Set up the MII Mgmt for a write cycle to the external PHY Control register (write the PHY address and Register address),<br>MIIMADD[0000_0000_0000_0000_0001_0001_0000_0000]<br>The control register (CR) is at offset address 0x00 from the external PHY address. (in this case 0x11) |
| Perform an MII Mgmt write cycle to the external PHY.<br>Write to MII Mgmt Control with 16-bit data intended for the external PHY Control register,<br>MIIMCON[0000_0000_0000_0000_0001_0010_0000_0000]<br>This enables the external PHY to restart Auto-Negotiations using the configuration set in the AN Advertisement register. |
| Check to see if MII Mgmt write is complete.<br>Read MII Mgmt Indicator register and check for Busy = 0,<br>MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000]<br>This indicates that the write cycle was completed. |
| Check to see if PHY has completed Auto-Negotiation.<br>Set up the MII Mgmt for a read cycle to the PHY MII Mgmt register (write the PHY address and Register address),<br>MIIMADD[0000_0000_0000_0000_0000_0010_0000_0001]<br>The PHY Status register is at address 0x1 and in this case the PHY Address is 0x2. |
| Perform an MII Mgmt read cycle of Status Register.<br>Clear MIIMCOM[Read Cycle]<br>Set MIIMCOM[Read Cycle]<br>(Uses the PHY address (2) and Register address (2) placed in MIIMADD register)<br>When MIIMIND[BUSY]=0,<br>read the MIIMSTAT register and check bit 10. (AN Done)<br>MIIMSTAT ---> [0000_0000_0000_0000_0000_0000_0010_0000]<br>Other information about the link is also returned. (Extend Status, No pre, Remote Fault, An Ability, Link status, extend Ability) |
| Perform an MII Mgmt read cycle of AN Expansion Register.<br>Setup MIIMADD[0000_0000_0000_0000_0001_0001_0000_0110]<br>Clear MIIMCOM[Read Cycle]<br>Set MIIMCOM[Read Cycle]<br>(Uses the PHY address (0x11) and Register address (6) placed in MIIMADD register)<br>When MIIMIND[BUSY]=0,<br>read the MII Mgmt AN Expansion register and check bits 13 and 14. (NP Able and Page Rx'd)<br>MII Mgmt AN Expansion ---> [0000_0000_0000_0000_0000_0000_0000_0110] |
| Perform an MII Mgmt read cycle of AN Link Partner Base Page Ability Register. (Optional)<br>Setup MIIMADD[0000_0000_0000_0000_0001_0001_0000_0101]<br>Clear MIIMCOM[Read Cycle]<br>Set MIIMCOM[Read Cycle]<br>(Uses the PHY address (0x11) and Register address (5) placed in MIIMADD register)<br>When MIIMIND[BUSY]=0,<br>read the MII Mgmt AN Link Partner Base Page Ability register and check bits 9 and 10. (Half and Full Duplex)<br>MII Mgmt AN Link Partner Base Page Ability ---> [0000_0000_0000_0000_0000_000x_1x10_0000] |

**Table 14-182. RGMII Mode Register Initialization Steps (continued)**

| |
|---|
| Clear IEVENT register,<br>IEVENT[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize IMASK (Optional)<br>IMASK[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize MACnADDR1/2 (Optional)<br>MACnADDR1/2[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize GADDR*n* (Optional)<br>GADDR*n*[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize RCTRL (Optional)<br>RCTRL[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize DMACTRL (Optional)<br>DMACTRL[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize (Empty) Transmit Descriptor ring and fill buffers with Data<br>Initialize TBASE0–TBASE7,<br>TBASE0–TBASE7[LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_L000] |
| Initialize (Empty) Receive Descriptor ring and fill with empty buffers<br>Initialize RBASE0–RBASE7,<br>RBASE0–RBASE7[LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_L000] |
| Enable Transmit Queues<br>Initialize TQUEUE |
| Enable Receive Queues<br>Initialize RQUEUE |
| Enable Rx and Tx,<br>MACCFG1[0000_0000_0000_0000_0000_0000_0000_0101] |

### 14.7.1.5 RMII Interface Mode

Table 14-183 shows the signals configurations required for RMII interface mode.

**Table 14-183. RMII Interface Mode Signal Configuration**

| eTSEC Signals | | | RMII Interface | | |
|---|---|---|---|---|---|
| | | | Frequency [MHz] 50 | | |
| | | | Voltage [V] 3.3 | | |
| **Signals** | **I/O** | **No. of Signals** | **Signals** | **I/O** | **No. of Signals** |
| GTX_CLK | O | 1 | leave unconnected | | |
| TX_CLK | I | 1 | REF_CLK | I | 1 |
| TxD[0] | O | 1 | TxD[0] | O | 1 |
| TxD[1] | O | 1 | TxD[1] | O | 1 |
| TxD[2] | O | 1 | leave unconnected | | |
| TxD[3] | O | 1 | leave unconnected | | |
| TxD[4] | O | 1 | leave unconnected | | |
| TxD[5] | O | 1 | leave unconnected | | |
| TxD[6] | O | 1 | leave unconnected | | |
| TxD[7] | O | 1 | leave unconnected | | |
| TX_EN | O | 1 | TX_EN | O | 1 |
| TX_ER | O | 1 | leave unconnected | | |
| RX_CLK | I | 1 | leave unconnected | | |
| RxD[0] | I | 1 | RxD[0] | I | 1 |
| RxD[1] | I | 1 | RxD[1] | I | 1 |
| RxD[2] | I | 1 | not used | | |
| RxD[3] | I | 1 | not used | | |
| RxD[4] | I | 1 | not used | | |
| RxD[5] | I | 1 | not used | | |
| RxD[6] | I | 1 | not used | | |
| RxD[7] | I | 1 | not used | | |
| RX_DV | I | 1 | CRS_DV | I | 1 |
| RX_ER | I | 1 | RX_ER | I | 1 |
| COL | I | 1 | not used | | |
| CRS | I | 1 | not used | | |
| **Sum** | | 25 | **Sum** | | 8 |

Table 14-184 describes the shared signals for the RMII interface.

**Table 14-184. Shared RMII Signals**

| eTSEC Signals | I/O | No. of Signals | GMII Signals | I/O | No. of Signals | Function |
|---|---|---|---|---|---|---|
| MDIO | I/O | 1 | MDIO | I/O | 1 | Management interface I/O |
| MDC | O | 1 | MDC | O | 1 | Management interface clock |
| TX_CLK | I | 1 | REF_CLK | I | 1 | Reference clock |
| **Sum** | | 3 | **Sum** | | 3 | |

Table 14-185 describes the register initializations required to configure the eTSEC in RMII mode.

**Table 14-185. RMII Mode Register Initialization Steps**

| |
|---|
| Set Soft_Reset,<br>MACCFG1[1000_0000_0000_0000_0000_0000_0000_0000] |
| Clear Soft_Reset,<br>MACCFG1[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize MACCFG2,<br>MACCFG2[0000_0000_0000_0000_0111_0010_0000_0101]<br>(I/F Mode = 2, Full Duplex = 1) |
| Initialize ECNTRL,<br>ECNTRL[0000_0000_0000_0000_0001_0000_0001_0000]<br>(Used to setup Reduced-Pin mode = 1, and TBIM = 0,statistics enable = 1) |
| Initialize MAC Station Address<br>MACSTNADDR2[0110_0000_0000_0010_0000_0000_0000_0000]<br>to 02608C:876543 for example |
| Initialize MAC Station Address<br>MACSTNADDR1[0100_0011_0110_0101_1000_0111_1000_1100]<br>to 02608C:876543 for example |
| Setup the MII Mgmt clock speed,<br>MIIMCFG[0000_0000_0000_0000_0000_0000_0000_1101]<br>set system clock divide by 14 for example to insure that MDC clock speed = 2.5 MHz |
| Read MII Mgmt Indicator register and check for Busy = 0,<br>MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000]<br>This indicates that the eTSEC MII Mgmt bus is idle. |
| Set up the MII Mgmt for a write cycle to external the PHY AN Advertisement register (write the PHY address and Register address),<br>MIIMADD[0000_0000_0000_0000_0001_0001_0000_0100]<br>The AN Advertisement register is at offset address 0x04 from the external PHY address. (in this case 0x11) |
| Perform an MII Mgmt write cycle to the external PHY.<br>Write to MII Mgmt Control with 16-bit data intended for the external PHY AN Advertisement register,<br>MIIMCON[0000_0000_0000_0000_u0uu_uuuu_uuuu_uuuu]<br>Where u must be selected by the user for proper system configuration. |
| Check to see if MII Mgmt write is complete.<br>Read MII Mgmt Indicator register and check for Busy = 0,<br>MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000]<br>This indicates that the write cycle was completed. |

### Table 14-185. RMII Mode Register Initialization Steps (continued)

| |
|---|
| Set up the MII Mgmt for a write cycle to the external PHY Control register (write the PHY address and Register address),<br>MIIMADD[0000_0000_0000_0000_0001_0001_0000_0000]<br>The control register is at offset address 0x00 from the external PHY address. (in this case 0x11) |
| Perform an MII Mgmt write cycle to the external PHY.<br>Write to MII Mgmt Control with 16-bit data intended for the external PHY Control register,<br>MIIMCON[0000_0000_0000_0000_0001_0010_0000_0000]<br>This enables the external PHY to restart Auto-Negotiations using the configuration set in the AN Advertisement register. |
| Check to see if MII Mgmt write is complete.<br>Read MII Mgmt Indicator register and check for Busy = 0,<br>MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000]<br>This indicates that the write cycle was completed. |
| Check to see if PHY has completed Auto-Negotiation.<br>Set up the MII Mgmt for a read cycle to the PHY MII Mgmt register (write the PHY address and Register address),<br>MIIMADD[0000_0000_0000_0000_0000_0010_0000_0001]<br>The PHY Status register is at address 0x1 and in this case the PHY Address is 0x2. |
| Perform an MII Mgmt read cycle of Status Register.<br>Clear MIIMCOM[Read Cycle]<br>Set MIIMCOM[Read Cycle]<br>(Uses the PHY address (2) and Register address (1) placed in MIIMADD register)<br>When MIIMIND[BUSY]=0,<br>read the MIIMSTAT register and check bit 10. (AN Done)<br>MIIMSTAT ---> [0000_0000_0000_0000_0000_0000_0010_0000]<br>Other information about the link is also returned. (Extend Status, No pre, Remote Fault, An Ability, Link status, extend Ability) |
| Perform an MII Mgmt read cycle of AN Expansion Register.<br>Setup MIIMADD[0000_0000_0000_0000_0001_0001_0000_0110]<br>Clear MIIMCOM[Read Cycle]<br>Set MIIMCOM[Read Cycle]<br>(Uses the PHY address (0x11) and Register address (6) placed in MIIMADD register)<br>When MIIMIND[BUSY]=0,<br>read the MII Mgmt AN Expansion register and check bits 13 and 14. (NP Able and Page Rx'd)<br>MII Mgmt AN Expansion ---> [0000_0000_0000_0000_0000_0000_0000_0110] |
| Perform an MII Mgmt read cycle of AN Link Partner Base Page Ability Register. (Optional)<br>Setup MIIMADD[0000_0000_0000_0000_0001_0001_0000_0101]<br>Clear MIIMCOM[Read Cycle]<br>Set MIIMCOM[Read Cycle]<br>(Uses the PHY address (0x11) and Register address (5) placed in MIIMADD register)<br>When MIIMIND[BUSY]=0,<br>read the MII Mgmt AN Link Partner Base Page Ability register and check bits 9 and 10. (Half and Full Duplex)<br>MII Mgmt AN Link Partner Base Page Ability ---> [0000_0000_0000_0000_0000_000x_x110_0000] |
| Setting up the MII Mgmt for a write cycle to TBI MII Mgmt register (write the TBI's address and Register address),<br>MIIMADD[0000_0000_0000_0000_0001_0000_0000_1011]<br>the TBI control register is at offset address 0x11 from TBIPA |
| Perform an MII Mgmt write cycle<br>Writing to MII Mgmt Control with 16-bit data intended for TBI's MII Mgmt control register (TBI control),<br>MIIMCON[0000_0000_0000_0000_0000_0010_0001_0000]<br>This configures the TBI control to GMII mode and AN sense |
| Check to see if MII Mgmt write is complete<br>Read MII Mgmt Indicator register and check for Busy = 0,<br>MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000]<br>This indicate that the write cycle was completed |

**Table 14-185. RMII Mode Register Initialization Steps (continued)**

| |
|---|
| Perform an MII Mgmt read cycle (0ptional)<br>Set MIIMCOM[Read Cycle]<br>(Uses the TBI address and Register address placed in MIIMADD register),<br>read the MIIMSTAT register and verify that<br>MIIMSTAT ---> [0000_0000_0000_0000_0000_0010_0001_0000] |
| Check to see if PHY has completed Auto-Negotiation<br>Setting up the MII Mgmt for a read cycle to PHY's MII Mgmt register (write the PHY's address and Register address),<br>MIIMADD[0000_0000_0000_0000_0000_0010_0000_0010]<br>the PHY Status control register is at address 0x2 and lets say the PHY Address is 0x2 |
| Perform an MII Mgmt read cycle of Status Register<br>Set MIIMCOM[Read Cycle]<br>(Uses the PHY address (2) and Register address (2) placed in MIIMADD register),<br>read the MIIMSTAT register and check bit 10 (AN Done)<br>MIIMSTAT ---> [0000_0000_0000_0000_0000_0000_0010_0000]<br>other information about the link is also returned (Extend Status, No pre, Remote Fault, An Ability, Link status, extend Ability) |
| Perform an MII Mgmt read cycle of AN Expansion Register<br>MIIMADD[0000_0000_0000_0000_0000_0010_0000_0110]<br>Set MIIMCOM[Read Cycle]<br>(Uses the PHY address (2) and Register address (6) placed in MIIMADD register),<br>read the MII Mgmt AN Expansion register and check bits 13 and 14 (NP Able and Page Rx'd)<br>MII Mgmt AN Expansion ---> [0000_0000_0000_0000_0000_0000_0000_0110] |
| Perform an MII Mgmt read cycle of AN Link Partner Base Page Ability Register (Optional)<br>MIIMADD[0000_0000_0000_0000_0000_0010_0000_0101]<br>Set MIIMCOM[Read Cycle]<br>(Uses the PHY address (2) and Register address (5) placed in MIIMADD register),<br>read the MII Mgmt AN Link Partner Base Page Ability register and check bits 9 and 10 (Half and Full Duplex)<br>MII Mgmt AN Link Partner Base Page Ability ---> [0000_0000_0000_0000_0000_00X_1110_0000] |
| Clear IEVENT register,<br>IEVENT[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize IMASK (Optional)<br>IMASK[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize GADDR*n* (Optional)<br>GADDR*n*[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize RCTRL (Optional)<br>RCTRL[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize DMACTRL (Optional)<br>DMACTRL[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize (Empty) Transmit Descriptor ring and fill buffers with Data<br>Initialize TBASE0–TBASE7,<br>TBASE0–TBASE7[LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_L000] |
| Initialize (Empty) Receive Descriptor ring and fill with empty buffers<br>Initialize RBASE0–RBASE7,<br>RBASE0–RBASE7[LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_L000] |
| Enable Transmit Queues<br>Initialize TQUEUE |
| Enable Receive Queues<br>Initialize RQUEUE |
| Enable Rx and Tx,<br>MACCFG1[0000_0000_0000_0000_0000_0000_0000_0101] |

### 14.7.1.6 RTBI Interface Mode

Table 14-186 describes the signal configurations required for RTBI interface mode.

**Table 14-186. RTBI Interface Mode Signal Configuration**

| eTSEC Signal s | | | RTBI Interface | | |
|---|---|---|---|---|---|
| | | | Frequency [MHz] 125 | | |
| | | | Voltage [V] 2.5 | | |
| **Signals** | **I/O** | **No. of Signals** | **Signals** | **I/O** | **No. of Signals** |
| GTX_CLK | O | 1 | GTX_CLK | O | 1 |
| TX_CLK | I | 1 | not used | | |
| TxD[0] | O | 1 | TCG[0]/TCG[5] | O | 1 |
| TxD[1] | O | 1 | TCG[1]/TCG[6] | O | 1 |
| TxD[2] | O | 1 | TCG[2]/TCG[7] | O | 1 |
| TxD[3] | O | 1 | TCG[3]/TCG[8] | O | 1 |
| TxD[4] | O | 1 | leave unconnected | | |
| TxD[5] | O | 1 | leave unconnected | | |
| TxD[6] | O | 1 | leave unconnected | | |
| TxD[7] | O | 1 | leave unconnected | | |
| TX_EN | O | 1 | TCG[4]/TCG[9] | O | 1 |
| TX_ER | O | 1 | leave unconnected | | |
| RX_CLK | I | 1 | RX_CLK | I | 1 |
| RxD[0] | I | 1 | RCG[0]/RCG[5] | I | 1 |
| RxD[1] | I | 1 | RCG[1]/RCG[6] | I | 1 |
| RxD[2] | I | 1 | RCG[2]/RCG[7] | I | 1 |
| RxD[3] | I | 1 | RCG[3]/RCG[8] | I | 1 |
| RxD[4] | I | 1 | not used | | |
| RxD[5] | I | 1 | not used | | |
| RxD[6] | I | 1 | not used | | |
| RxD[7] | I | 1 | not used | | |
| RX_DV | I | 1 | RCG[4]/RCG[9] | I | 1 |
| RX_ER | I | 1 | not used | | |
| COL | I | 1 | not used | | |
| CRS | I | 1 | not used | | |
| **Sum** | | 25 | **sum** | | 12 |

Table 14-187 describes the shared signals for the RTBI interface.

**Table 14-187. Shared RTBI Signals**

| eTSEC Signals | I/O | No. of Signals | GMII Signals | I/O | No. of Signals | Function |
|---|---|---|---|---|---|---|
| MDIO | I/O | 1 | MDIO | I/O | 1 | Management interface I/O |
| MDC | O | 1 | MDC | O | 1 | Management interface clock |
| ECGTX_CLK125 | I | 1 | GTX_CLK125 | I | 1 | Reference clock |
| Sum |  | 3 | Sum |  | 3 |  |

Table 14-188 describes the register initializations required to configure the eTSEC in RTBI mode.

**Table 14-188. RTBI Mode Register Initialization Steps**

|  |
|---|
| Set Soft_Reset,<br>MACCFG1[1000_0000_0000_0000_0000_0000_0000_0000] |
| Clear Soft_Reset,<br>MACCFG1[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize MACCFG2,<br>MACCFG2[0000_0000_0000_0000_0111_0010_0000_0101]<br>(I/F Mode = 2, Full Duplex = 1) |
| Initialize ECNTRL,<br>ECNTRL[0000_0000_0000_0000_0001_0000_0000_0000]<br>(This example has Statistics Enable = 1) |
| Initialize MAC Station Address,<br>MACSTNADDR2[0110_0000_0000_0010_0000_0000_0000_0000]<br>to 02608C:876543, for example. |
| Initialize MAC Station Address,<br>MACSTNADDR1[0100_0011_0110_0101_1000_0111_1000_1100]<br>to 02608C:876543, for example. |
| Assign a Physical address to the TBI,<br>TBIPA[0000_0000_0000_0000_0000_0000_0001_0000]<br>set to 16, for example. |
| Setup the MII Mgmt clock speed,<br>MIIMCFG[0000_0000_0000_0000_0000_0000_0000_0101]<br>Set source clock divide by 14, for example, to insure that TSEC_MDC clock speed is not greater than 2.5 MHz. |
| Read MII Mgmt Indicator register and check for Busy = 0,<br>MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000]<br>This indicates that the eTSEC MII Mgmt bus is idle. |
| Set up the MII Mgmt for a read cycle to TBI’s Control register (write the TBI’s address and Register address),<br>MIIMADD[0000_0000_0000_0000_0001_0000_0000_0000]<br>The control register (CR) is at offset address 0x0 from TBIPA. |

**Table 14-188. RTBI Mode Register Initialization Steps (continued)**

| |
|---|
| Perform an MII Mgmt read cycle to verify state of TBI Control Register(0ptional)<br>Clear MIIMCOM[Read Cycle]<br>Set MIIMCOM[Read Cycle]<br>(Uses the TBI address and Register address placed in MIIMADD register),<br>When MIIMIND[BUSY]=0,<br>read the MIIMSTAT and look for AN Enable and other bit information. |
| Set up the MII Mgmt for a write cycle to TBI’s AN Advertisement register (write the PHY address and Register address),<br>MIIMADD[0000_0000_0000_0000_0001_0000_0000_0100]<br>The AN Advertisement register is at offset address 0x04 from the TBI’s address. (in this case 0x10) |
| Perform an MII Mgmt write cycle to TBI.<br>Write to MII Mgmt Control with 16-bit data intended for TBI’s AN Advertisement register,<br>MIIMCON[0000_0000_0000_0000_0000_0001_1010_0000]<br>This advertises to the Link Partner that the TBI supports PAUSE and Full Duplex mode and does not support Half Duplex mode. |
| Check to see if MII Mgmt write is complete.<br>Read MII Mgmt Indicator register and check for Busy = 0,<br>MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000]<br>This indicates that the write cycle was completed. |
| Set up the MII Mgmt for a write cycle to TBI’s Control register (write the PHY address and Register address),<br>MIIMADD[0000_0000_0000_0000_0001_0000_0000_0000]<br>The control register (CR) is at offset address 0x00 from the TBI’s address. (in this case 0x10) |
| Perform an MII Mgmt write cycle to TBI.<br>Writing to MII Mgmt Control with 16-bit data intended for TBI’s Control register,<br>MIIMCON[0000_0000_0000_0000_0001_0010_0000_0000]<br>This enables the TBI to restart Auto-Negotiations using the configuration set in the AN Advertisement register. |
| Check to see if MII Mgmt write is complete.<br>Read MII Mgmt Indicator register and check for Busy = 0,<br>MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000]<br>This indicates that the write cycle was completed. |
| Check to see if PHY has completed Auto-Negotiation.<br>Set up the MII Mgmt for a read cycle to the PHY MII Mgmt register (write the PHY address and Register address),<br>MIIMADD[0000_0000_0000_0000_0001_0000_0000_0001]<br>The PHY Status control register is at address 0x1 and in this case the PHY Address is 0x10. |
| Perform an MII Mgmt read cycle of Status Register.<br>Clear MIIMCOM[Read Cycle]<br>Set MIIMCOM[Read Cycle]<br>(Uses the PHY address (2) and Register address (2) placed in MIIMADD register),<br>When MIIMIND[BUSY]=0,<br>read the MIIMSTAT register and check bit 10 (AN Done)<br>MIIMSTAT ---> [0000_0000_0000_0000_0000_0000_0010_0000]<br>Other information about the link is also returned. (Extend Status, No pre, Remote Fault, An Ability, Link status, extend Ability) |

**Table 14-188. RTBI Mode Register Initialization Steps (continued)**

| |
|---|
| Perform an MII Mgmt read cycle of AN Expansion Register.<br>Setup MIIMADD[0000_0000_0000_0000_0001_0000_0000_0110]<br>Clear MIIMCOM[Read Cycle]<br>Set MIIMCOM[Read Cycle]<br>(Uses the PHY address (0x10) and Register address (6) placed in MIIMADD register),<br>When MIIMIND[BUSY]=0,<br>read the MII Mgmt AN Expansion register and check bits 13 and 14. (NP Able and Page Rx'd)<br>MII Mgmt AN Expansion ---> [0000_0000_0000_0000_0000_0000_0000_0110] |
| Perform an MII Mgmt read cycle of AN Link Partner Base Page Ability Register. (Optional)<br>Setup MIIMADD[0000_0000_0000_0000_0001_0000_0000_0101]<br>Clear MIIMCOM[Read Cycle]<br>Set MIIMCOM[Read Cycle]<br>(Uses the PHY address (0x10) and Register address (5) placed in MIIMADD register),<br>When MIIMIND[BUSY]=0,<br>read the MII Mgmt AN Link Partner Base Page Ability register and check bits 9 and 10. (Half and Full Duplex)<br>MII Mgmt AN Link Partner Base Page Ability ---> [0000_0000_0000_0000_0000_00x_x110_0000] |
| Clear IEVENT register,<br>IEVENT[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize IMASK (Optional)<br>IMASK[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize MACnADDR1/2 (Optional)<br>MACnADDR1/2[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize GADDR*n* (Optional)<br>GADDR*n*[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize RCTRL (Optional)<br>RCTRL[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize DMACTRL (Optional)<br>DMACTRL[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize (Empty) Transmit Descriptor ring and fill buffers with Data<br>Initialize TBASE0–TBASE7,<br>TBASE0–TBASE7[LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_L000] |
| Initialize (Empty) Receive Descriptor ring and fill with empty buffers<br>Initialize RBASE0–RBASE7,<br>RBASE0–RBASE7[LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_L000] |
| Enable Transmit Queues<br>Initialize TQUEUE |
| Enable Receive Queues<br>Initialize RQUEUE |
| Enable Rx and Tx,<br>MACCFG1[0000_0000_0000_0000_0000_0000_0000_0101] |

### 14.7.1.7 8-Bit FIFO Mode

Table 14-189 describes the signal configurations required for 8-bit FIFO interface mode.

**Table 14-189. 8-Bit FIFO Interface Mode Signal Configurations**

| **eTSEC Signals** | | | **8-Bit FIFO Interface** | | |
|---|---|---|---|---|---|
| | | | **Frequency [MHz] 155** | | |
| | | | **Voltage [V] 2.5** | | |
| **Signals** | **I/O** | **No. of Signals** | **Signals** | **I/O** | **No. of Signals** |
| TSEC*n*_GTX_CLK | O | 1 | FIFO*n*_GTX_CLK | O | 1 |
| TSEC*n*_TX_CLK | I | 1 | FIFO*n*_TX_CLK | I | 1 |
| TSEC*n*_TxD[0] | O | 1 | FIFO*n*_TxD[0] | O | 1 |
| TSEC*n*_TxD[1] | O | 1 | FIFO*n*_TxD[1] | O | 1 |
| TSEC*n*_TxD[2] | O | 1 | FIFO*n*_TxD[2] | O | 1 |
| TSEC*n*_TxD[3] | O | 1 | FIFO*n*_TxD[3] | O | 1 |
| TSEC*n*_TxD[4] | O | 1 | FIFO*n*_TxD[4] | O | 1 |
| TSEC*n*_TxD[5] | O | 1 | FIFO*n*_TxD[5] | O | 1 |
| TSEC*n*_TxD[6] | O | 1 | FIFO*n*_TxD[6] | O | 1 |
| TSEC*n*_TxD[7] | O | 1 | FIFO*n*_TxD[7] | O | 1 |
| TSEC*n*_TX_EN | O | 1 | FIFO*n*_TX_EN | O | 1 |
| TSEC*n*_TX_ER | O | 1 | FIFO*n*_TX_ER | O | 1 |
| TSEC*n*_RX_CLK | I | 1 | FIFO*n*_RX_CLK | I | 1 |
| TSEC*n*_RxD[0] | I | 1 | FIFO*n*_RxD[0] | I | 1 |
| TSEC*n*_RxD[1] | I | 1 | FIFO*n*_RxD[1] | I | 1 |
| TSEC*n*_RxD[2] | I | 1 | FIFO*n*_RxD[2] | I | 1 |
| TSEC*n*_RxD[3] | I | 1 | FIFO*n*_RxD[3] | I | 1 |
| TSEC*n*_RxD[4] | I | 1 | FIFO*n*_RxD[4] | I | 1 |
| TSEC*n*_RxD[5] | I | 1 | FIFO*n*_RxD[5] | I | 1 |
| TSEC*n*_RxD[6] | I | 1 | FIFO*n*_RxD[6] | I | 1 |
| TSEC*n*_RxD[7] | I | 1 | FIFO*n*_RxD[7] | I | 1 |
| TSEC*n*_RX_DV | I | 1 | FIFO*n*_RX_DV | I | 1 |
| TSEC*n*_RX_ER | I | 1 | FIFO*n*_RX_ER | I | 1 |
| TSEC*n*_COL | I | 1 | FIFO*n*_TX_FC | I | 1 |
| TSEC*n*_CRS | I/O | 1 | FIFO*n*_RX_FC | O | 1 |
| MDIO | I/O | 1 | leave unconnected | | |
| MDC | O | 1 | leave unconnected | | |
| **Sum** | | 27 | **Sum** | | 25 |

Table 14-190 describes the register initializations required to configure the eTSEC in 8-bit FIFO mode.

**Table 14-190. 8-Bit FIFO Mode Register Initialization Steps**

| |
|---|
| Set FIFO Soft_Reset,<br>FIFOCFG[0000_0000_0000_0000_1100_0000_0000_0000]<br>(Reset RX = 1, reset Tx = 1, Rx enable = 0, Tx enable = 0) |
| Clear FIFO Soft_Reset,<br>FIFOCFG[0000_0000_0000_0000_0000_0000_0000_1000]<br>(Reset RX = 0, reset Tx = 0, Rx enable = 0, Tx enable = 0) |
| Ensure MACCFG2 is set to default values.<br>MACCFG2[0000_0000_0000_0000_0111_0000_0000_0000] |
| Initialize ECNTRL,<br>ECNTRL[0000_0000_0000_0000_1000_0000_0000_0000]<br>(Used to set up FIFO mode = 1, and statistics enable = 0) |
| Clear IEVENT register,<br>IEVENT[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize IMASK (Optional)<br>IMASK[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize RCTRL (Optional)<br>RCTRL[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize DMACTRL (Optional)<br>DMACTRL[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize (Empty) transmit descriptor ring and fill buffers with data<br>Initialize TBASE0–TBASE7,<br>TBASE0–TBASE7[LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_L000] |
| Initialize (Empty) Receive Descriptor ring and fill with empty buffers<br>Initialize RBASE0–RBASE7,<br>RBASE0–RBASE7[LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_L000] |
| Enable Transmit Queues<br>Initialize TQUEUE |
| Enable Receive Queues<br>Initialize RQUEUE |
| Enable Rx and Tx over FIFO,<br>FIFOCFG[0000_0000_0000_0000_0011_0000_1101_1000]<br>(Rx enable = 1, Tx enable = 1, enable flow control and CRC, 8-bit mode) |

### 14.7.1.8 SGMII Interface Support

**Table 14-191. SGMII Interface Signal Configuration (4-Wire)**

| **SerDes Signals** | | | **SGMII Interface** | | |
|---|---|---|---|---|---|
| **Frequency [MHz] 1250** | | | **Frequency [MHz] 1250** | | |
| **Voltage [V] LVDS** | | | **Voltage [V] LVDS** | | |
| **Signals** | **I/O** | **No. of Signals** | **Signals** | **I/O** | **No. of Signals** |
| TX*n*/$\overline{\text{TX}n}$ | O | 2 | TXD | O | 2 |
| RX*n*/$\overline{\text{RX}n}$ | I | 2 | RXD | I | 2 |
| **Sum** | | 4 | **Sum** | | 4 |

SGMII mode initialization sequence is very similar to TBI mode initialization. Additional initialization is required for the SerDes. An example of SGMII mode initialization sequence is shown in Table 14-192.

**NOTE**

SGMII mode utilizes the internal TBI PHY. The internal TBI PHY only auto-negotiates at 1 Gbps. However, 10 Mbps and 100 Mbps speeds are supported in SGMII mode. It is recommended that the external PHY inform the MAC if the desired link speed is not 1 Gbps. Software can perform MII management cycles to determine the external PHY link speed and program ECNTRL and MACCFG2 accordingly.

**Table 14-192. SGMII Mode Register Initialization Steps**

| *Initialize SerDes to select SGMII. The initialization sequence should be prepended with SerDes initialization.* |
|---|
| Set Soft_Reset,<br>MACCFG1[1000_0000_0000_0000_0000_0000_0000_0000] |
| Clear Soft_Reset,<br>MACCFG1[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize MACCFG2,<br>MACCFG2[0000_0000_0000_0000_0111_0010_0000_0101]<br>(I/F Mode = 2, Full Duplex = 1)<br>(Set I/F mode = 1 in SGMII 10/100 Mbps speed) |
| Initialize ECNTRL,<br>ECNTRL[0000_0000_0000_0000_0001_0000_0010_0010]<br>(This example has Statistics Enable = 1, TBIM = 1, SGMIIM = 1)<br>(Set R100M = 1 in SGMII 100 Mbps speed) |
| Initialize MAC Station Address<br>MACSTNADDR2[0110_0000_0000_0010_0000_0000_0000_0000]<br>to 02608C:876543, for example. |
| Initialize MAC Station Address<br>MACSTNADDR1[0100_0011_0110_0101_1000_0111_1000_1100]<br>to 02608C:876543, for example. |

**Table 14-192. SGMII Mode Register Initialization Steps (continued)**

| |
|---|
| Assign a Physical address to the TBI,<br>TBIPA[0000_0000_0000_0000_0000_0000_0001_0000]<br>set to 16, for example. |
| Setup the MII Mgmt clock speed,<br>MIIMCFG[0000_0000_0000_0000_000_0000_0000_0101]<br>set source clock divide by 14 for example to insure that MDC clock speed is not greater than 2.5 MHz |
| Read MII Mgmt Indicator register and check for Busy = 0,<br>MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000]<br>This indicates that the eTSEC MII Mgmt bus is idle. |
| Set up the MII Mgmt for a read cycle to TBI’s Control register (write the PHY address and Register address),<br>MIIMADD[0000_0000_0000_0000_0001_0000_0000_0000]<br>the control register (CR) is at offset address 0x00 from the TBI’s address. |
| Perform an MII Mgmt read cycle to verify state of TBI Control Register (optional)<br>Clear MIIMCOM[Read Cycle]<br>Set MIIMCOM[Read Cycle]<br>(Uses the TBI address and Register address placed in MIIMADD register),<br>When MIIMIND[BUSY] = 0,<br>read the MIIMSTAT and look for AN Enable and other bit information. |
| Set up the MII Mgmt for a write cycle to TBICON register (write the PHY address and Register address),<br>MIIMADD[0000_0000_0000_0000_0001_0000_0001_0001]<br>The TBICON register is at offset address 0x11 from the TBI’s address. |
| Perform an MII Mgmt write cycle to TBI.<br>Writing to MII Mgmt Control with 16-bit data intended for TBICON register,<br>MIIMCON[0000_0000_0000_0000_0000_0000_0010_0000]<br>This sets TBI in single clock mode and MII Mode off to enable communication with SerDes. |
| Check to see if MII Mgmt write is complete.<br>Read MII Mgmt Indicator register and check for Busy = 0,<br>MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000]<br>This indicates that the write cycle was completed. |
| Set up the MII Mgmt for a write cycle to TBI’s AN Advertisement register (write the PHY address and Register address),<br>MIIMADD[0000_0000_0000_0000_0001_0000_0000_0100]<br>The AN Advertisement register is at offset address 0x04 from the TBI’s address. |
| Perform an MII Mgmt write cycle to TBI.<br>Writing to MII Mgmt Control with 16-bit data intended for TBI’s AN Advertisement register,<br>MIIMCON[0000_0000_0000_0000_0000_0001_1010_0000]<br>This advertises to the Link Partner that the TBI supports PAUSE and Full Duplex mode and does not support Half Duplex mode. |
| Check to see if MII Mgmt write is complete.<br>Read MII Mgmt Indicator register and check for Busy = 0,<br>MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000]<br>This indicates that the write cycle was completed. |
| *Additional SerDes setup as required* |
| Set up the MII Mgmt for a write cycle to TBI’s Control register (write the PHY address and Register address),<br>MIIMADD[0000_0000_0000_0000_0001_0000_0000_0000]<br>the control register (CR) is at offset address 0x00 from the TBI’s address. |

**Table 14-192. SGMII Mode Register Initialization Steps (continued)**

| |
|---|
| Perform an MII Mgmt write cycle to TBI.<br>Writing to MII Mgmt Control with 16-bit data intended for TBI's Control register,<br>MIIMCON[0000_0000_0000_0000_0001_0011_0100_0000]<br>This enables the TBI to restart Auto-Negotiations using the configuration set in the AN Advertisement register. |
| Check to see if MII Mgmt write is complete.<br>Read MII Mgmt Indicator register and check for Busy = 0,<br>MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000]<br>This indicates that the write cycle was completed. |
| Check to see if PHY has completed Auto-Negotiation.<br>Set up the MII Mgmt for a read cycle to PHY MII Mgmt register (write the PHY address and Register address),<br>MIIMADD[0000_0000_0000_0000_0001_0000_0000_0001]<br>The PHY Status control register is at address 0x1 and in this case the PHY Address is 0x10. |
| Perform an MII Mgmt read cycle of Status Register.<br>Clear MIIMCOM[Read Cycle]<br>Set MIIMCOM[Read Cycle]<br>(Uses the PHY address (2) and Register address (2) placed in MIIMADD register),<br>When MIIMIND[BUSY] = 0,<br>read the MIIMSTAT register and check bit 10 (AN Done)<br>MII Mgmt AN Expansion ---> [0000_0000_0000_0000_0000_0000_0000_0110]<br>Other information about the link is also returned. (Extend Status, No pre, Remote Fault, An Ability, Link status, extend Ability) |
| Perform an MII Mgmt read cycle of AN Expansion Register.<br>Setup MIIMADD[0000_0000_0000_0000_0001_0000_0000_0110]<br>Clear MIIMCOM[Read Cycle]<br>Set MIIMCOM[Read Cycle]<br>(Uses the PHY address (0x10) and Register address (6) placed in MIIMADD register),<br>When MIIMIND[BUSY] = 0,<br>read the MII Mgmt AN Expansion register and check bits 13 and 14 (NP Able and Page Rx'd)<br>MII Mgmt AN Expansion ---> [0000_0000_0000_0000_0000_0000_0000_0110] |
| Perform an MII Mgmt read cycle of AN Link Partner Base Page Ability Register. (Optional)<br>Setup MIIMADD[0000_0000_0000_0000_0001_0000_0000_0101]<br>Clear MIIMCOM[Read Cycle]<br>Set MIIMCOM[Read Cycle]<br>(Uses the PHY address (0x10) and Register address (5) placed in MIIMADD register),<br>When MIIMIND[BUSY] = 0,<br>read the MII Mgmt AN Link Partner Base Page Ability register and check bits 9 and 10. (Half and Full Duplex)<br>MII Mgmt AN Link Partner Base Page Ability ---> [0000_0000_0000_0000_0000_000x_1110_0000] |
| Clear IEVENT register,<br>IEVENT[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize IMASK (Optional)<br>IMASK[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize MACnADDR1/2 (Optional)<br>MACnADDR1/2[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize GADDR*n* (Optional)<br>GADDR*n*[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize RCTRL (Optional)<br>RCTRL[0000_0000_0000_0000_0000_0000_0000_0000] |
| Initialize DMACTRL (Optional)<br>DMACTRL[0000_0000_0000_0000_0000_0000_0000_0000] |

**Table 14-192. SGMII Mode Register Initialization Steps (continued)**

| |
|---|
| Initialize (Empty) Transmit Descriptor ring and fill buffers with Data<br>Initialize TBASE0–TBASE7,<br>TBASE0–TBASE7[LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_L000] |
| Initialize (Empty) Receive Descriptor ring and fill with empty buffers<br>Initialize RBASE0–RBASE7,<br>RBASE0–RBASE7[LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_L000] |
| Enable Transmit Queues<br>Initialize TQUEUE |
| Enable Receive Queues<br>Initialize RQUEUE |
| Enable Rx and Tx,<br>MACCFG1[0000_0000_0000_0000_0000_0000_0000_0101] |

# Chapter 15
# DMA Controller

This chapter describes the DMA controller offered on this device.

## 15.1 Introduction

The DMA controller transfers blocks of data between the many interface and functional blocks of this device, independent of the  core or external hosts.

### 15.1.1 Block Diagram

Figure 15-1 shows the block diagram of the DMA controller.

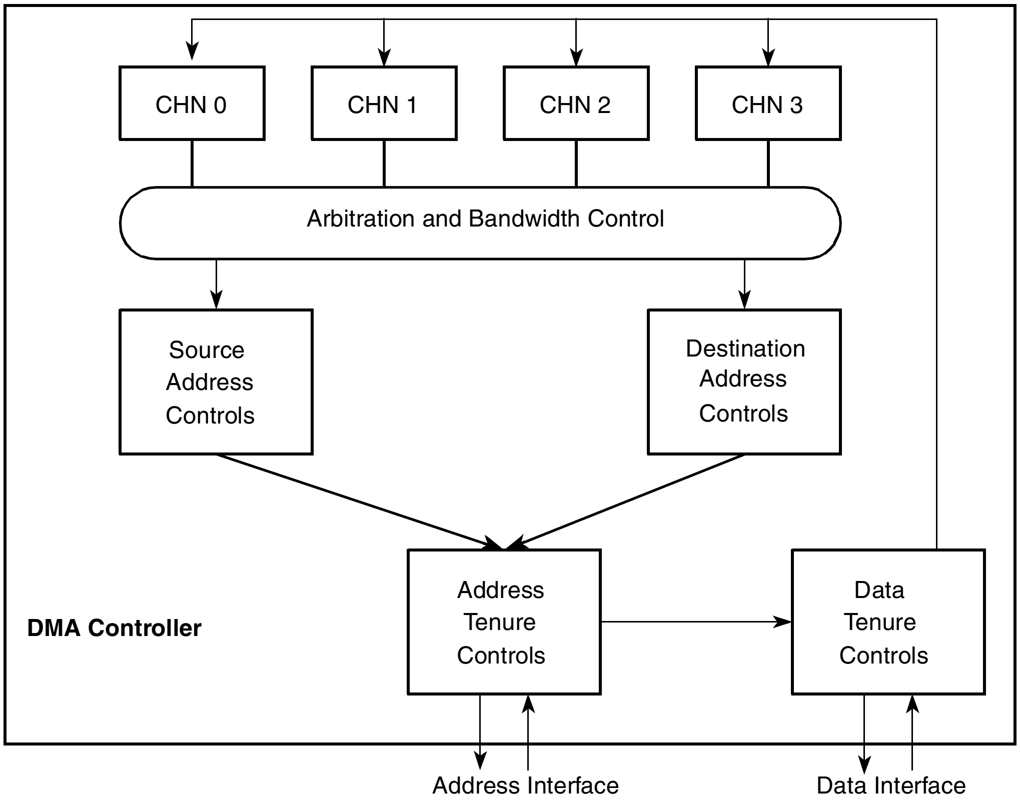


**Figure 15-1. DMA Block Diagram**

### 15.1.2 Overview

The DMA controller has four high-speed DMA channels. Both the  core and external devices can initiate DMA transfers. All channels are capable of complex data movement and advanced transaction chaining. Figure 15-1 is a high-level block diagram of the DMA controller. Operations such as descriptor fetches and block transfers are initiated by each channel. A channel is selected by the arbitration logic and information is passed to the source and destination control blocks for processing. The source and destination blocks generate read and write requests to the address tenure engine, which manages the DMA master port address interface. After a transaction is accepted by the master port, control is transferred to the data tenure engine that manages the read and write data transfers. A channel remains active in the shared resources for the duration of the data transfer unless the allotted bandwidth per channel is reached.

### 15.1.3 Features

The DMA controller offers the following features:

- Four high-speed/high-bandwidth channels accessible by local and remote masters
- Basic DMA operation modes (direct, simple chaining)
- Extended DMA operation modes (advanced chaining and stride capability)
- Cascading descriptor chains
- Misaligned transfers
- Programmable bandwidth control between channels
- Three priority levels supported for source and destination transactions
- Interrupt on error and completed segment, list, or link
- Externally-controlled transfer using $\overline{\mathrm{DMA\_DREQ}}$, $\overline{\mathrm{DMA\_DACK}}$, and $\overline{\mathrm{DMA\_DDONE}}$

### 15.1.4 Modes of Operation

The DMA block has two modes of operation: basic and extended. Basic mode is the DMA legacy mode, which does not support advanced features. Extended mode supports advanced features such as striding and flexible descriptor structures.

These two basic modes allow users to initiate and end DMA transfers in various ways. Table 15-1 summarizes the relationship between the modes and the following features:

- Direct mode. No descriptors are involved. Software must initialize the required fields as described in Table 15-1 before starting a transfer.
- Chaining mode. Software must initialize descriptors in memory and the required fields as described in Table 15-1 before starting a transfer.
- Single-write start mode. The DMA process can be started using a single-write command to either the descriptor address register in one of the chaining modes or the source/destination address registers in one of the direct modes.
- External control capability. This allows an external agent to start, pause, and check the status of a DMA transfer which has already been initialized.

- Channel continue capability. The channel continue capability allows software the flexibility of having the DMA controller start with descriptors that have already been programmed while software continues to build more descriptors in memory.
- Channel abort capability. The software can abort a previously initiated transfer by setting the bit MR*n*[CA]. The DMA controller terminates all outstanding transfers initiated by the channel without generating any errors before entering an idle state.

**Table 15-1. Relationship of Modes and Features**

| Mode | Mode with One Additional Feature | Mode with Two Additional Features |
|---|---|---|
| B (Basic) | BD (basic direct) | BDS (BD single-write start) |
| | | BDE (BD external control) |
| | BC (basic chaining) | BCE (BC external control) |
| | | BCS (BC single-write start) |
| Ext (Extended) | ExtD (extended direct) | ExtDS (ExtD single-write start) |
| | | ExtDE (ExtD external control) |
| | ExtC (extended chaining) | ExtCE (ExtC external control) |
| | | ExtCS (ExtC single-write start) |

Table 15-2 describes bit settings required for each DMA mode of operation.

**Table 15-2. DMA Mode Bit Settings**

| Modes with Features | MR*n*[XFE] | MR*n*[CTM] | MR*n*[SRW] | MR*n*[CDSM/SWSM] | MR*n*[EMS_EN] |
|---|---|---|---|---|---|
| Basic Direct Modes | | | | | |
| Basic direct | 0 | 1 | 0 | 0 | 0 |
| Basic direct external control | 0 | 1 | 0 | 0 | 1 |
| Basic direct single-write start | 0 | 1 | 1 | 1 or 0 | 0 |
| Basic Chaining Modes | | | | | |
| Basic chaining | 0 | 0 | Reserved | 0 | 0 |
| Basic chaining external control | 0 | 0 | Reserved | 0 | 1 |
| Basic chaining single-write start | 0 | 0 | Reserved | 1 | 0 |
| Extended Direct Modes | | | | | |
| Extended direct | 1 | 1 | 0 | 0 | 0 |
| Extended direct external control | 1 | 1 | 0 | 0 | 1 |
| Extended direct single-write start | 1 | 1 | 1 | 1 or 0 | 0 |

**Table 15-2. DMA Mode Bit Settings (continued)**

| Modes with Features | MR*n*[XFE] | MR*n*[CTM] | MR*n*[SRW] | MR*n*[CDSM/SWSM] | MR*n*[EMS_EN] |
|---|---|---|---|---|---|
| Extended Chaining Modes | | | | | |
| Extended chaining | 1 | 0 | Reserved | 0 | 0 |
| Extended chaining external control | 1 | 0 | Reserved | 0 | 1 |
| Extended chaining single-write start | 1 | 0 | Reserved | 1 | 0 |

Refer to Section 15.4, “Functional Description,” for details on these modes.

Figure 15-2 shows the general DMA operational flow chart.

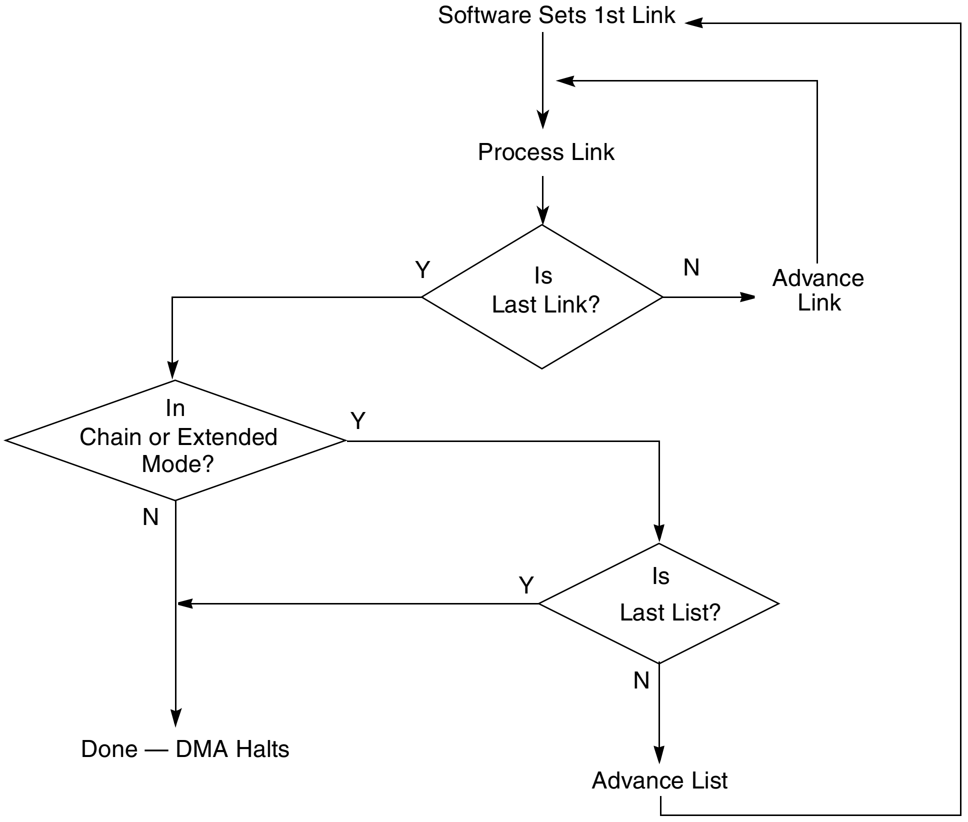


**Figure 15-2. DMA Operational Flow Chart**

## 15.2 External Signal Description

This section describes the DMA signals.

### 15.2.1 Signal Overview

Figure 15-3 summarizes the DMA controller signals.

**Figure 15-3. DMA Signal Summary**

### 15.2.2 Detailed Signal Descriptions

Table 15-3 describes the DMA signals.

**Table 15-3. DMA Signals—Detailed Signal Descriptions**

| Signal | I/O | Description | |
|---|---|---|---|
| DMA_DREQ*n*<br>DMA request | I | DMA request. Indicates the start of a DMA transfer or a restart from a paused request. Assertion of DMA_DREQ*n* causes MR*n*[CS] to be set, thereby activating the corresponding DMA channel. | |
| | | **State Meaning** | Asserted—Assertion of DMA_DREQ*n* while DMA_DACK*n* is negated causes a new transfer to start OR resumes a paused transfer if the EMP_EN bit is set. Assertion while DMA_DACK*n* is asserted results in an illegal condition.<br>Negated—Negation while DMA_DACK*n* is asserted has no effect. Negation before the assertion of DMA_DACK*n* results in an illegal condition. |
| | | **Timing** | Assertion—Can be asserted asynchronously<br>Negation— Must remain asserted at least until the assertion of the corresponding DMA_DACK*n* |
| DMA_DACK*n* | O | DMA acknowledge. Indicates that a DMA transfer is currently in progress | |
| | | **State Meaning** | Asserted—Indicates that a DMA transfer is currently in progress. Asserted after the assertion of DMA_DREQ*n* to indicate the start of a transfer<br>Negated—Negated after finishing a complete transfer or after entering a paused state if MR*n*[EMP_EN] is set |
| | | **Timing** | Assertion—Asynchronous assertion; asserted for more than three system clocks<br>Negation—Asynchronous negation; negated for more than three system clocks |
| DMA_DDONE*n* | O | DMA done. Indicates that a DMA transfer is complete | |
| | | **State Meaning** | Asserted—Indicates transfer completion. SRn[CB] is clear. Note, however, that write data may still be queued at the target interface or in the process of transfer on an external interface.<br>Negated—Indicates that the current transfer is in process |
| | | **Timing** | Assertion—Always asserts asynchronously after the negation of the final DMA_DACK*n* to indicate completion of a transfer. For a paused transfer, DMA_DDONE*n* is asserted asynchronously after the negation of the final DMA_DACK*n*.<br>Negation—Negated asynchronously after the assertion of DMA_DREQ*n* for the next transfer |

## 15.3 Memory Map/Register Definition

This section provides a detailed description of all accessible DMA memory and registers. The descriptions include individual bit level descriptions and reset states of each register. Undefined 4-byte address spaces within offset 0x000–0xFFF are reserved.

Table 15-4 lists the DMA registers and their offsets. Note that the full register address is comprised of the programmable CCSRBAR together with the fixed DMA block base address and offset listed in Table 15-4.

In this table and in the register figures and field descriptions, the following access definitions apply:

- Reserved fields are always ignored for the purposes of determining access type.
- R/W, R, and W (read/write, read only, and write only) indicate that all the non-reserved fields in a register have the same access type.
- w1c indicates that all of the non-reserved fields in a register are cleared by writing ones to them.

- Mixed indicates a combination of access types.
- Special is used when no other category applies. In this case the register figure and field description table should be read carefully.

**Table 15-4. DMA Register Summary**

| Offset | Register | Access | Reset | Section/Page |
|---|---|---|---|---|
| **DMA Controller Block Base Address: 0x2_1000** | | | | |
| 0x100 | MR0—DMA 0 mode register | R/W | 0x0000_0000 | 15.3.1.1/15-9 |
| 0x104 | SR0—DMA 0 status register | Mixed | 0x0000_0000 | 15.3.1.2/15-11 |
| 0x108 | ECLNDAR0—DMA 0 current link descriptor extended address register | R/W | 0x0000_0000 | 15.3.1.3/15-13 |
| 0x10C | CLNDAR0—DMA 0 current link descriptor address register | R/W | 0x0000_0000 | 15.3.1.3/15-13 |
| 0x110 | SATR0—DMA 0 source attributes register | R/W | 0x0000_0000 | 15.3.1.4/15-15 |
| 0x114 | SAR0—DMA 0 source address register | R/W | 0x0000_0000 | 15.3.1.5/15-15 |
| 0x118 | DATR0—DMA 0 destination attributes register | R/W | 0x0000_0000 | 15.3.1.6/15-16 |
| 0x11C | DAR0—DMA 0 destination address register | R/W | 0x0000_0000 | 15.3.1.7/15-17 |
| 0x120 | BCR0—DMA 0 byte count register | R/W | 0x0000_0000 | 15.3.1.8/15-18 |
| 0x124 | ENLNDAR0—DMA 0 next link descriptor extended address register | R/W | 0x0000_0000 | 15.3.1.9/15-18 |
| 0x128 | NLNDAR0—DMA 0 next link descriptor address register | R/W | 0x0000_0000 | 15.3.1.9/15-18 |
| 0x130 | ECLSDAR0—DMA 0 current list descriptor extended address register | R/W | 0x0000_0000 | 15.3.1.10/15-19 |
| 0x134 | CLSDAR0—DMA 0 current list descriptor address register | R/W | 0x0000_0000 | 15.3.1.10/15-19 |
| 0x138 | ENLSDAR0—DMA 0 next list descriptor extended address register | R/W | 0x0000_0000 | 15.3.1.11/15-21 |
| 0x13C | NLSDAR0—DMA 0 next list descriptor address register | Mixed | 0x0000_0000 | 15.3.1.11/15-21 |
| 0x140 | SSR0—DMA 0 source stride register | R/W | 0x0000_0000 | 15.3.1.12/15-22 |
| 0x144 | DSR0—DMA 0 destination stride register | R/W | 0x0000_0000 | 15.3.1.13/15-22 |
| 0x148–<br>0x17C | Reserved | — | — | — |
| 0x180 | MR1—DMA 1 mode register | R/W | 0x0000_0000 | 15.3.1.1/15-9 |
| 0x184 | SR1—DMA 1 status register | Mixed | 0x0000_0000 | 15.3.1.2/15-11 |
| 0x188 | ECLNDAR1—DMA 1 current link descriptor extended address register | R/W | 0x0000_0000 | 15.3.1.3/15-13 |
| 0x18C | CLNDAR1—DMA 1 current link descriptor address register | R/W | 0x0000_0000 | 15.3.1.3/15-13 |
| 0x190 | SATR1—DMA 1 source attributes register | R/W | 0x0000_0000 | 15.3.1.4/15-15 |
| 0x194 | SAR1—DMA 1 source address register | R/W | 0x0000_0000 | 15.3.1.5/15-15 |
| 0x198 | DATR1—DMA 1 destination attributes register | R/W | 0x0000_0000 | 15.3.1.6/15-16 |
| 0x19C | DAR1—DMA 1 destination address register | R/W | 0x0000_0000 | 15.3.1.7/15-17 |
| 0x1A0 | BCR1—DMA 1 byte count register | R/W | 0x0000_0000 | 15.3.1.8/15-18 |

Table 15-4. DMA Register Summary (continued)

| Offset | Register | Access | Reset | Section/Page |
|---|---|---|---|---|
| 0x1A4 | ENLNDAR1—DMA 1 next link descriptor extended address register | R/W | 0x0000_0000 | 15.3.1.9/15-18 |
| 0x1A8 | NLNDAR1—DMA 1 next link descriptor address register | R/W | 0x0000_0000 | 15.3.1.9/15-18 |
| 0x1B0 | ECLSDAR1—DMA 1 current list descriptor extended address register | R/W | 0x0000_0000 | 15.3.1.10/15-19 |
| 0x1B4 | CLSDAR1—DMA 1 current list descriptor address register | R/W | 0x0000_0000 | 15.3.1.10/15-19 |
| 0x1B8 | ENLSDAR1—DMA 1 next list descriptor extended address register | R/W | 0x0000_0000 | 15.3.1.11/15-21 |
| 0x1BC | NLSDAR1—DMA 1 next list descriptor address register | R/W | 0x0000_0000 | 15.3.1.11/15-21 |
| 0x1C0 | SSR1—DMA 1 source stride register | R/W | 0x0000_0000 | 15.3.1.12/15-22 |
| 0x1C4 | DSR1—DMA 1 destination stride register | R/W | 0x0000_0000 | 15.3.1.13/15-22 |
| 0x1C8–<br>0x1FC | Reserved | — | — | — |
| 0x200 | MR2—DMA 2 mode register | R/W | 0x0000_0000 | 15.3.1.1/15-9 |
| 0x204 | SR2—DMA 2 status register | Mixed | 0x0000_0000 | 15.3.1.2/15-11 |
| 0x208 | ECLNDAR2—DMA 2 current link descriptor extended address register | R/W | 0x0000_0000 | 15.3.1.3/15-13 |
| 0x20C | CLNDAR2—DMA 2 current link descriptor address register | R/W | 0x0000_0000 | 15.3.1.3/15-13 |
| 0x210 | SATR2—DMA 2 source attributes register | R/W | 0x0000_0000 | 15.3.1.4/15-15 |
| 0x214 | SAR2—DMA 2 source address register | R/W | 0x0000_0000 | 15.3.1.5/15-15 |
| 0x218 | DATR2—DMA 2 destination attributes register | R/W | 0x0000_0000 | 15.3.1.6/15-16 |
| 0x21C | DAR2—DMA 2 destination address register | R/W | 0x0000_0000 | 15.3.1.7/15-17 |
| 0x220 | BCR2—DMA 2 byte count register | R/W | 0x0000_0000 | 15.3.1.8/15-18 |
| 0x224 | ENLNDAR2—DMA 2 next link descriptor extended address register | R/W | 0x0000_0000 | 15.3.1.9/15-18 |
| 0x228 | NLNDAR2—DMA 2 next link descriptor address register | R/W | 0x0000_0000 | 15.3.1.9/15-18 |
| 0x230 | ECLSDAR2—DMA 2 current list descriptor extended address register | R/W | 0x0000_0000 | 15.3.1.10/15-19 |
| 0x234 | CLSDAR2—DMA 2 current list descriptor address register | R/W | 0x0000_0000 | 15.3.1.10/15-19 |
| 0x238 | ENLSDAR2—DMA 2 next list descriptor extended address register | R/W | 0x0000_0000 | 15.3.1.11/15-21 |
| 0x23C | NLSDAR2—DMA 2 next list descriptor address register | R/W | 0x0000_0000 | 15.3.1.11/15-21 |
| 0x240 | SSR2—DMA 2 source stride register | R/W | 0x0000_0000 | 15.3.1.12/15-22 |
| 0x244 | DSR2—DMA 2 destination stride register | R/W | 0x0000_0000 | 15.3.1.13/15-22 |
| 0x248–<br>0x27C | Reserved | — | — | — |
| 0x280 | MR3—DMA 3 mode register | R/W | 0x0000_0000 | 15.3.1.1/15-9 |
| 0x284 | SR3—DMA 3 status register | Mixed | 0x0000_0000 | 15.3.1.2/15-11 |

Table 15-4. DMA Register Summary (continued)

| Offset | Register | Access | Reset | Section/Page |
|---|---|---|---|---|
| 0x288 | ECLNDAR3—DMA 3 current link descriptor extended address register | R/W | 0x0000_0000 | 15.3.1.3/15-13 |
| 0x28C | CLNDAR3—DMA 3 current link descriptor address register | R/W | 0x0000_0000 | 15.3.1.3/15-13 |
| 0x290 | SATR3—DMA 3 source attributes register | R/W | 0x0000_0000 | 15.3.1.4/15-15 |
| 0x294 | SAR3—DMA 3 source address register | R/W | 0x0000_0000 | 15.3.1.5/15-15 |
| 0x298 | DATR3—DMA 3 destination attributes register | R/W | 0x0000_0000 | 15.3.1.6/15-16 |
| 0x29C | DAR3—DMA 3 destination address register | R/W | 0x0000_0000 | 15.3.1.7/15-17 |
| 0x2A0 | BCR3—DMA 3 byte count register | R/W | 0x0000_0000 | 15.3.1.8/15-18 |
| 0x2A4 | ENLNDAR3—DMA 3 next link descriptor extended address register | R/W | 0x0000_0000 | 15.3.1.9/15-18 |
| 0x2A8 | NLNDAR3—DMA 3 next link descriptor address register | R/W | 0x0000_0000 | 15.3.1.9/15-18 |
| 0x2B0 | ECLSDAR3—DMA 3 current list descriptor extended address register | R/W | 0x0000_0000 | 15.3.1.10/15-19 |
| 0x2B4 | CLSDAR3—DMA 3 current list descriptor address register | R/W | 0x0000_0000 | 15.3.1.10/15-19 |
| 0x2B8 | ENLSDAR3—DMA 3 next list descriptor extended address register | R/W | 0x0000_0000 | 15.3.1.11/15-21 |
| 0x2BC | NLSDAR3—DMA 3 next list descriptor address register | R/W | 0x0000_0000 | 15.3.1.11/15-21 |
| 0x2C0 | SSR3—DMA 3 source stride register | R/W | 0x0000_0000 | 15.3.1.12/15-22 |
| 0x2C4 | DSR3—DMA 3 destination stride register | R/W | 0x0000_0000 | 15.3.1.13/15-22 |
| 0x2C8–0x2FC | Reserved | — | — | — |
| 0x300 | DGSR—DMA general status register | R | 0x0000_0000 | 15.3.1.14/15-23 |

## 15.3.1 DMA Register Descriptions

The following sections describe the DMA registers. The majority of these registers are channel-specific and can be identified by one of the four offsets that describe the register.

#### 15.3.1.1 Mode Registers (MR*n*)

The mode register allows software to start a DMA transfer and to control various DMA transfer characteristics. Figure 15-4 describes the MR*n*.

Offset 0x100
0x180
0x200
0x280

Access: Read/Write

| | 0–3 | 4–7 | 8–9 | 10 | 11–12 | 13 | 14–15 |
|---|---|---|---|---|---|---|---|
| R<br>W | — | BWC | — | EMP_EN | — | EMS_EN | DAHTS |
| Reset | All zeros | | | | | | |

| | 16–17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | SAHTS | DAHE | SAHE | — | SRW | EOSIE | EOLNIE | EOLSIE | EIE | XFE | CDSM/SWSM | CA | CTM | CC | CS |
| Reset | All zeros | | | | | | | | | | | | | | |

**Figure 15-4. DMA Mode Registers (MR*n*)**

Table 15-5 describes the MR*n* fields.

**Table 15-5. MR*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–3 | — | Reserved |
| 4–7 | BWC | Bandwidth/pause control. Defined when single and multiple channels are executing or if MR*n*[EMP_EN] is set in external transfer mode.<br>The value of MR*n*[BWC] determines how many bytes a given channel is allowed to transfer before the DMA engine pauses the current channel and re-arbitrates (switches to the next channel).<br>In external pause mode, the value of MR*n*[BWC] dictates how many bytes are allowed to transfer before pausing the channel, after which a new assertion of $\overline{\text{DREQ}}$ resumes channel operation.<br>0000 1 byte<br>0001 2 bytes<br>0010 4 bytes<br>0011 8 bytes<br>0100 16 bytes<br>0101 32 bytes<br>0110 64 bytes<br>0111 128 bytes<br>1000 256 bytes<br>1001 512 bytes<br>1010 1024 bytes<br>1011–1110 Reserved<br>1111 Disable bandwidth sharing to allow uninterrupted transfers from each channel. |
| 8–9 | — | Reserved |
| 10 | EMP_EN | External master pause enable. Valid only if MR*n*[EMS_EN] is set.<br>0 Disable the external master pause feature.<br>1 Enable the external master pause feature. Channel is paused as described by MR*n*[BWC]. |
| 11–12 | — | Reserved |
| 13 | EMS_EN | External master start enable. This bit does not apply to single-write start modes (direct or chaining).<br>0 Disable the channel from being started by an external DMA start pin.<br>1 Enable the channel to be started by an external DMA start pin, which sets MR*n*[CS]. |

### Table 15-5. MR*n* Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 14–15 | DAHTS | Destination address hold transfer size. Indicates the transfer size used for each transaction while MR*n*[DAHE] is set. The byte count register must be in multiples of the size and the destination address register must be aligned based on the size. The transfer size assigned to MR*n*[DAHTS] must be equal to or smaller than that assigned to MR*n*[BWC] to avoid undefined behavior.<br>00 1 byte<br>01 2 bytes<br>10 4 bytes<br>11 8 bytes |
| 16–17 | SAHTS | Source address hold transfer size. Indicates the transfer size used for each transaction while MR*n*[SAHE] is set. The byte count register must be in multiples of the size and the source address register must be aligned based on the size. The transfer size assigned to MR*n*[SAHTS] must be equal to or smaller than that assigned to MR*n*[BWC] to avoid undefined behavior.<br>00 1 byte<br>01 2 bytes<br>10 4 bytes<br>11 8 bytes |
| 18 | DAHE | Destination address hold enable<br>0 Disable destination address hold<br>1 Enable the DMA controller to hold the destination address of a transfer to the size specified by MR*n*[DAHTS]. Hardware only supports aligned transfers for this feature. |
| 19 | SAHE | Source address hold enable<br>0 Disable source address hold<br>1 Enable the DMA controller to hold the source address of a transfer to the size specified by MR*n*[SAHTS]. Hardware only supports aligned transfers for this feature. |
| 20 | — | Reserved |
| 21 | SRW | Single register write (Direct mode only; reserved for chaining mode.)<br>0 Normal operation<br>1 Enable a write to the source address register to simultaneously set MR*n*[CS], starting a DMA transfer, when MR*n*[CDSM/SWSM] is also set. Setting this bit and clearing CDSM/SWSM causes a write to the destination address register to simultaneously set MR*n*[CS], starting a DMA transfer. |
| 22 | EOSIE | End-of-segments interrupt enable<br>0 Do not generate an interrupt at the completion of a data transfer. CLNDAR*n*[EOSIE] overrides this bit on a link descriptor basis.<br>1 Generate an interrupt at the completion of a data transfer (That is, SR*n*[EOSI] is set). This bit overrides the CLNDAR*n*[EOSIE]. |
| 23 | EOLNIE | End-of-links interrupt enable<br>0 Do not generate an interrupt at the completion of a list of DMA transfers.<br>1 Generate an interrupt at the completion of a list of DMA transfers (That is, NLNDAR*n*[EOLND] is set). |
| 24 | EOLSIE | End-of-lists interrupt enable<br>0 Do not generate an interrupt at the completion of all DMA transfers.<br>1 Generate an interrupt at the completion of all DMA transfers (That is, NLNDAR*n*[EOLND] and NLSDAR*n*[EOLSD] are set). |
| 25 | EIE | Error interrupt enable<br>0 Do not generate an interrupt if a programming or transfer error is detected.<br>1 Generate an interrupt if a programming or transfer error is detected. |

Table 15-5. MR*n* Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 26 | XFE | Extended features enable<br>0 Disable the new chaining features.<br>1 Enable the new chaining features. |
| 27 | CDSM/<br>SWSM | • In chaining mode: current descriptor start mode/single-write start mode<br>—In basic mode (MR*n*[XFE] is cleared), setting this bit causes a write to the current link descriptor address register to simultaneously set MR*n*[CS], starting a DMA transfer.<br>—In extended chaining mode (MR*n*[XFE] is set), setting this bit causes a write to the current list descriptor address register to simultaneously set MR*n*[CS], starting a DMA transfer.<br>• In direct mode: Setting this bit and MR*n*[SRW] causes a write to the source address register to simultaneously set MR*n*[CS], starting a DMA transfer. Clearing this bit and setting MR*n*[SRW] causes a write to the destination address register to simultaneously set MR*n*[CS], starting a DMA transfer. This bit must be cleared when MR*n*[SRW] is cleared. |
| 28 | CA | Channel abort<br>0 No effect<br>1 Cause the current transfer to be aborted and SR*n*[CB] to be cleared if the channel is busy. The channel remains in the idle state until a new transfer is programmed. |
| 29 | CTM | Channel transfer mode<br>0 Configure the channel in chaining mode.<br>1 Configure the channel into direct mode. This means that software is responsible for placing all the required parameters into necessary registers to start the DMA process. |
| 30 | CC | Channel continue. This bit applies only to chaining mode and is cleared by hardware after the first descriptor read when continuing a transfer. This bit is reserved for external master mode.<br>0 No effect<br>1 The DMA transfer restarts the transferring process starting at the current descriptor address. |
| 31 | CS | Channel start. This bit is also automatically set by hardware during single-write start mode and external master start enable mode. Note that in external control mode, deasserting $\overline{\text{DMA\_DREQ}}$ does NOT clear this bit.<br>0 Halt the DMA process if channel is busy (SR*n*[CB] is set). No effect if the channel is not busy.<br>1 Start the DMA process if channel is not busy (CB is cleared). If the channel was halted (CS = 0 and CB = 1), the transfer continues from the point at which it was halted. |

## 15.3.1.2 Status Registers (SR*n*)

The status registers, shown in Figure 15-5, report various DMA conditions during and after a DMA transfer.

Offset 0x104
0x184
0x204
0x284

Access: Mixed

| | 0–23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|
| R | — | TE | — | CH | PE | EOLNI | CB | EOSI | EOLSI |
| W | | w1c | | | w1c | w1c | | w1c | w1c |
| Reset | All zeros | | | | | | | | |

Figure 15-5. Status Registers (SR*n*)

Table 15-6 describes the bits of the SR*n*.

**Table 15-6. SR*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–23 | — | Reserved |
| 24 | TE | Transfer error (Bit reset, write 1 to clear)<br>0 No error condition during the DMA transfer<br>1 Error condition during the DMA transfer. See Section 15.4.3, “DMA Errors,” for additional information. |
| 25 | — | Reserved |
| 26 | CH | Channel halted. Cleared automatically by hardware if MR*n*[CS] is set again for resuming a halted transfer<br>0 Channel is not halted. If software attempts to halt an idle channel (SR*n*[CB] is cleared), this bit remains 0.<br>1 DMA transfer was successfully halted by software and can be resumed. |
| 27 | PE | Programming error (bit reset, write 1 to clear)<br>0 No programming error detected<br>1 A programming error is detected that prevents the DMA transfer from occurring. |
| 28 | EOLNI | End-of-links interrupt. After transferring the last block of data in the last link descriptor, if MR*n*[EOLSIE] is set, then this bit is set and an interrupt is generated.<br>(Bit reset, write 1 to clear) |
| 29 | CB | Channel busy<br>0 DMA transfer is finished, an error occurred, or a channel abort occurred.<br>1 DMA transfer is currently in progress. |
| 30 | EOSI | End-of-segment interrupt. In chaining mode, after finishing a data transfer, if MR*n*[EOSIE] is set or if CLNDAR*n*[EOSIE] is set, this bit gets set and an interrupt is generated. In direct mode, if MRn[EOSIE] is set, this bit gets set and an interrupt is generated.<br>(Bit reset, write 1 to clear) |
| 31 | EOLSI | End-of-list interrupt. After transferring the last block of data in the last list descriptor, if MR*n*[EOLSIE] is set, then this bit is set and an interrupt is generated.<br>(Bit reset, write 1 to clear) |

### 15.3.1.3 Current Link Descriptor Address Registers (CLNDAR*n* and ECLNDAR*n*)

Current link descriptor address registers contain the address of the current link descriptor. In basic chaining mode, shown in Figure 15-6, software must initialize these registers to point to the first link descriptors in memory.

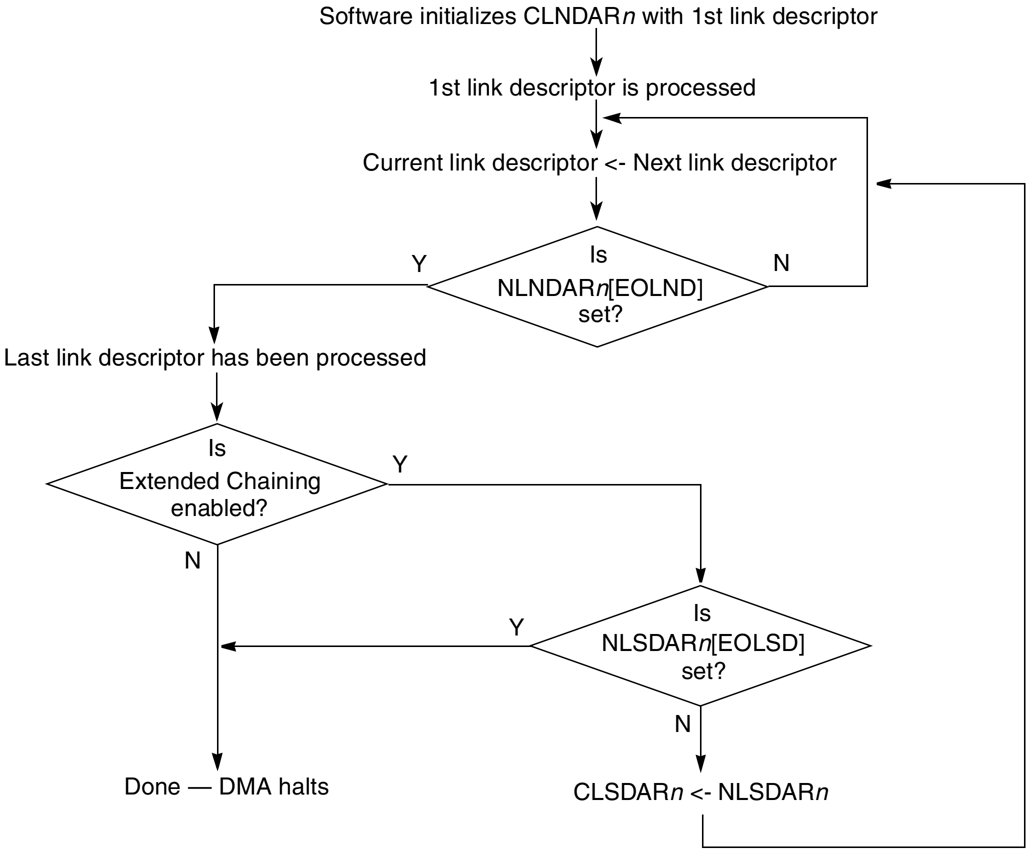

Software initializes CLNDARn with 1st link descriptor
1st link descriptor is processed
Current link descriptor <- Next link descriptor
Is NLNDARn[EOLND] set?
Y
N
Last link descriptor has been processed
Is Extended Chaining enabled?
Y
N
Is NLSDARn[EOLSD] set?
Y
N
Done — DMA halts
CLSDARn <- NLSDARn

**Figure 15-6. Basic Chaining Mode Flow Chart**

After the current descriptor is processed, the current link descriptor address register is loaded from the next link descriptor address registers and NLNDAR*n*[EOLND] in the next link descriptor address register is examined. If EOLND is zero, the DMA controller reads in the new current link descriptor for processing. If EOLND is set, the last descriptor of the list was just completed. If extended chaining mode is not enabled, all DMA transfers are complete and the DMA controller halts.

If extended chaining mode is enabled, the DMA controller examines the state of NLSDAR*n*[EOLSD] in the next list descriptor address register. If EOLSD is clear, the controller loads the contents of the next list descriptor address register into the current list descriptor address register and reads the new list descriptor from memory. If EOLSD is set, all DMA transfers are complete and the DMA controller halts.

Figure 15-7 shows ECLNDAR*n*.

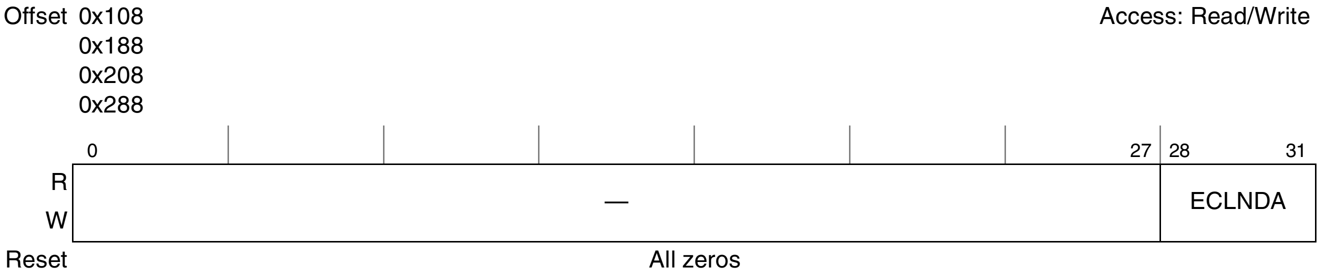


**Figure 15-7. Extended Current Link Descriptor Address Registers (ECLNDAR*n*)**

**Table 15-7. ECLNDAR*n* Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 0–27 | — | Reserved |
| 28–31 | ECLNDA | Current link descriptor extended address (upper 4 bits of 36-bit address) |

Figure 15-8 shows CLNDAR*n*.

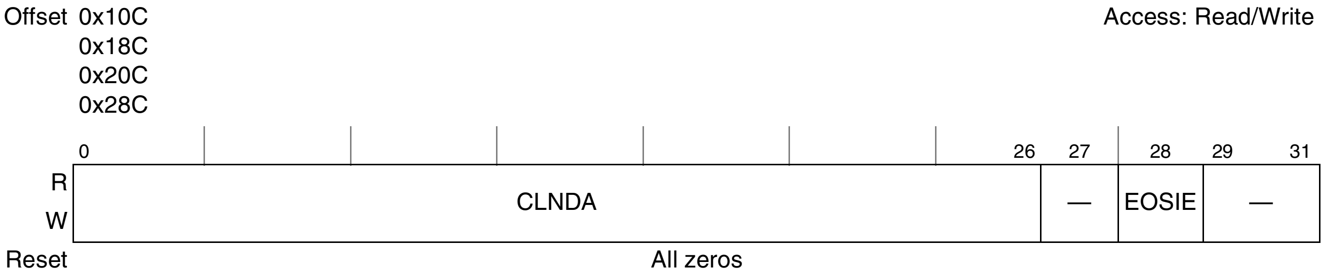


**Figure 15-8. Current Link Descriptor Address Registers (CLNDAR*n*)**

Table 15-8 describes the fields of the CLNDAR*n*.

**Table 15-8. CLNDAR*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–26 | CLNDA | Current link descriptor address. Contains the current descriptor address of the buffer descriptor in memory. The descriptor must be aligned to a 32-byte boundary. (This is the lower portion of the 36-bit address formed by CLNDAR*n*[CLNDA] and ECLNDAR*n*[ECLNDA].) |
| 27 | — | Reserved |
| 28 | EOSIE | End-of-segment interrupt enable<br>0 Do not generate an interrupt upon completion of the current DMA transfer for the current link descriptor.<br>1 Generate an interrupt upon completion of the current DMA transfer for the current link descriptor. |
| 29–31 | — | Reserved |

### 15.3.1.4 Source Attributes Registers (SATR*n*)

The source attributes registers, shown in Figure 15-9, contain the transaction attributes to be used for the DMA operation. Stride mode is enabled by setting SATR*n*[SSME].

Offset 0x110<br>0x190<br>0x210<br>0x290

Access: Read/Write

| | 0–6 | 7 | 8–11 | 12–15 | 16–27 | 28–31 |
|---|---|---|---|---|---|---|
| R<br>W | — | SSME | — | SREADTTYPE | — | ESAD |
| Reset | All zeros | | | | | |

**Figure 15-9. Source Attributes Registers (SATR*n*)**

Table 15-9 describes the fields of the SATR*n*.

**Table 15-9. SATR*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–6 | — | Reserved |
| 7 | SSME | Source stride mode enable<br>0 Stride mode disabled<br>1 Stride mode enabled<br>Ignored in basic mode (MR*n*[XFE] is cleared). Striding on the source address can be accomplished by enabling SATR*n*[SSME] and setting the desired stride size and distance in the SSR*n*. |
| 8–11 | — | Reserved |
| 12–15 | SREADTTYPE | DMA source transaction type. Reserved values result in a programming error being detected and logged in SR[PE].<br>Transaction type to run on local address space<br>0000–0001 Reserved<br>0011 Reserved<br>0100 Read, don't snoop local processor<br>0101 Read, snoop local processor<br>0111–1111 Reserved |
| 16–27 | — | Reserved |
| 28–31 | ESAD | Extended source address.<br>ESAD represents the four high-order bits of the 36-bit source address. |

### 15.3.1.5 Source Address Registers (SAR*n*)

The source address registers, shown in Figure 15-10, contain the address from which the DMA controller reads data. In direct mode, if MR*n*[CDSM/SWSM] and MR*n*[SRW] are set, a write to this register simultaneously sets MR*n*[CS], starting a DMA transfer. Software must ensure that this is a valid address.

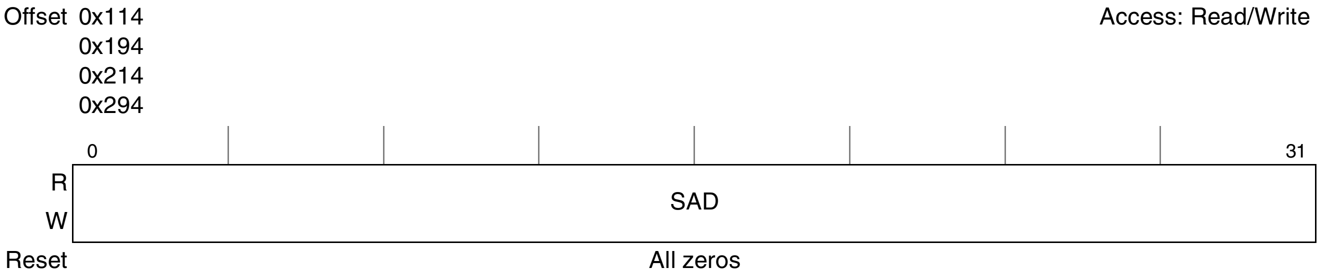


**Figure 15-10. Source Address Registers (SAR*n*)**

Table 15-10 describes the field of the SAR*n*.

**Table 15-10. SAR*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–31 | SAD | Source address. This register contains the low-order bits of the 36-bit source address of the DMA transfer. The contents are updated after every DMA write operation unless the final stride of a striding operation is less than the stride size, in which case it remains equal to the address from which the last stride began. |

### 15.3.1.6 Destination Attributes Registers (DATR*n*)

The destination attributes registers, shown in Figure 15-11, contain the transaction attributes for the DMA operation. Stride mode is enabled by setting DATR*n*[DSME].

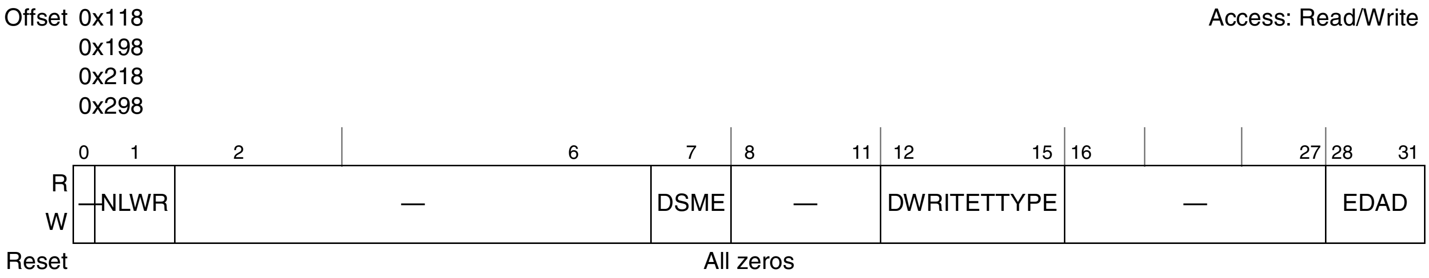


**Figure 15-11. Destination Attributes Registers (DATR*n*)**

Table 15-11 describes the fields of the DATR*n*.

**Table 15-11. DATR*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | — | Reserved |
| 1 | NLWR | No last write with response. Not valid for flush transaction type.<br>0 Last write for transfer is a write with target response.<br>1 Last write for transfer is a write without target response. |
| 2–6 | — | Reserved |

Table 15-11. DATR*n* Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 7 | DSME | Destination stride mode enable<br>0 Stride mode disabled<br>1 Stride mode enabled<br>Ignored in basic mode (MR*n*[XFE] is cleared). Striding on the destination address can be accomplished by setting DSME and setting the desired stride size and distance in DSR*n*. |
| 8–11 | — | Reserved |
| 12–15 | DWRITETTYPE | DMA destination transaction type. Reserved values result in a programming error being detected and logged in SR[PE].Transaction type to run on local address space<br>0000–0011 Reserved<br>0100 Write, don't snoop local processor<br>0101 Write, snoop local processor<br>0110<br>0111<br>1000–1111 Reserved |
| 16–27 | — | Reserved |
| 28–31 | EDAD | Extended destination address. EDAD represents the four high-order bits of the 36-bit destination address. |

### 15.3.1.7 Destination Address Registers (DAR*n*)

The destination address registers, shown in Figure 15-12, contain the addresses to which the DMA controller writes data.

In direct mode, if MR*n*[SRW] is set and MR*n*[CDSM/SWSM] is cleared, a write to this register simultaneously sets MR*n*[CS], starting a DMA transfer. Software must ensure that this is a valid address.

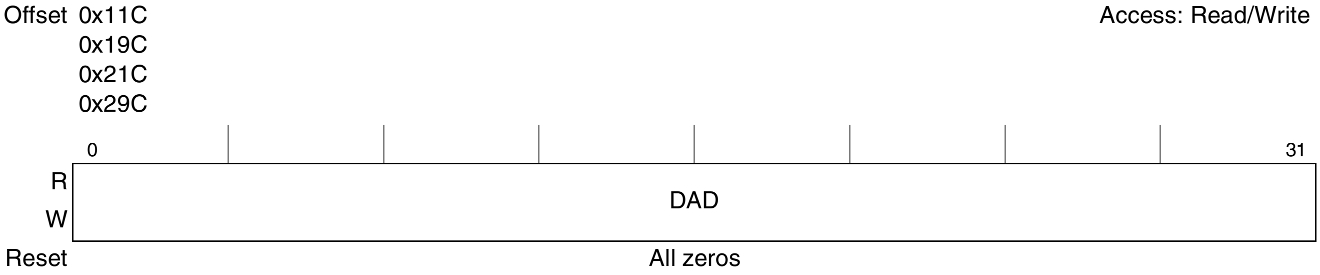


Figure 15-12. Destination Address Registers (DAR*n*)

Table 15-12 describes the field of the DAR*n*.

Table 15-12. DAR*n* Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–31 | DAD | Destination address. This register contains the destination address of the DMA transfer. The contents are updated after every DMA write operation unless the final stride of a striding operation is less than the stride size, in which case it remains equal to the address from which the last stride began. |

### 15.3.1.8 Byte Count Registers (BCR*n*)

The byte count register, shown in Figure 15-13, contains the number of bytes to transfer.

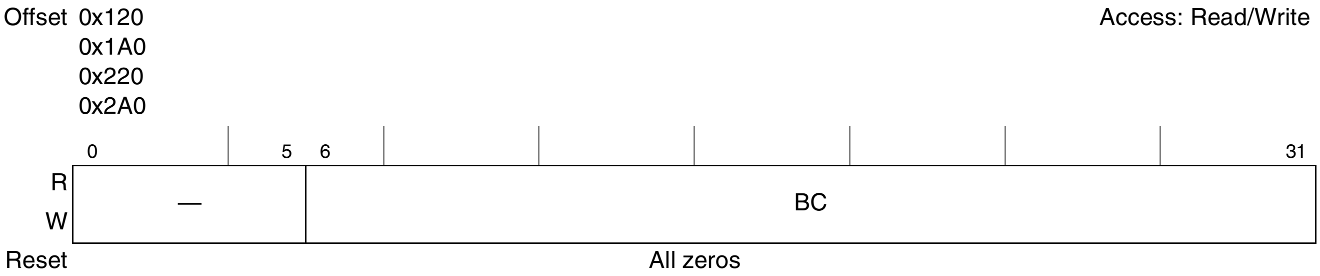


**Figure 15-13. Byte Count Registers (BCR*n*)**

Table 15-13 describes the fields of the BCR*n*.

**Table 15-13. BCR*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–5 | — | Reserved |
| 6–31 | BC | Byte count. Contains the number of bytes to transfer. The value in this register is decremented after each DMA read operation. The maximum transfer size is $(2^{26}) – 1$ bytes. |

### 15.3.1.9 Next Link Descriptor Address Registers (NLNDAR*n* and ENLNDAR*n*)

The next link descriptor address registers, shown in Figure 15-14 and Figure 15-15, contain the address for the next link descriptor in memory. Contents transferred to the current descriptor address registers become effective for the current transfer in basic and extended chaining modes.

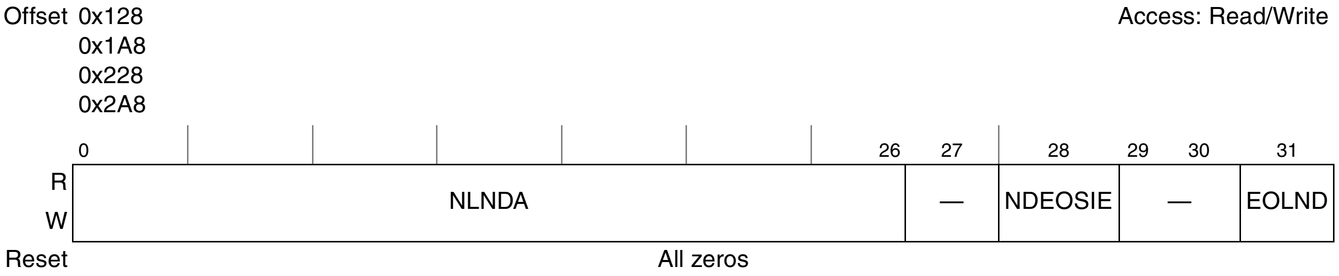


**Figure 15-14. Next Link Descriptor Address Registers (NLNDAR*n*)**

Table 15-14 describes the fields of the NLNDAR*n* registers.

**Table 15-14. NLNDAR*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–26 | NLNDA | Next link descriptor address. Contains the next link descriptor address in memory. The descriptor must be aligned to a 32-byte boundary. |
| 27 | — | Reserved |

**Table 15-14. NLNDAR*n* Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 28 | NDEOSIE | Next descriptor end-of-segment interrupt enable<br>0 Do not generate an interrupt if the current DMA transfer for the current descriptor is finished.<br>1 Generate an interrupt if the current DMA transfer for the current descriptor is finished. |
| 29–30 | — | Reserved |
| 31 | EOLND | End-of-links descriptor. This bit is ignored in direct mode.<br>0 This descriptor is not the last link descriptor in memory for this list.<br>1 This descriptor is the last link descriptor in memory for this list. If this bit is set, the DMA controller advances to the next list descriptor in memory if NLSDAR*n*[EOLSD] is also set in extended mode. |

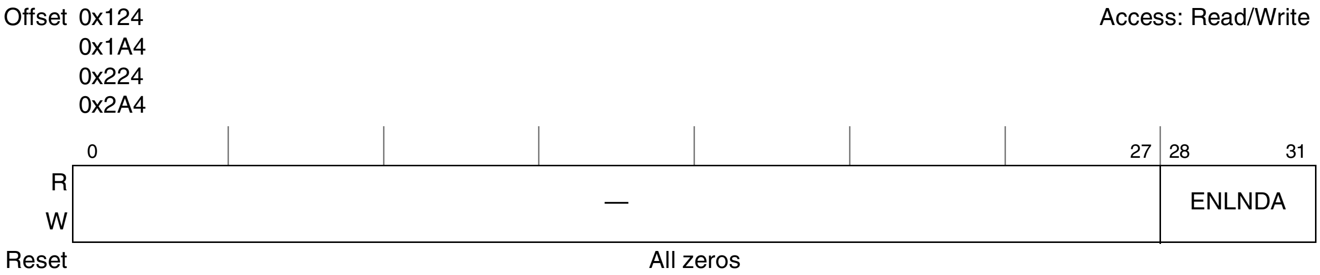


**Figure 15-15. Extended Next Link Descriptor Address Registers (ENLNDAR*n*)**

Table 15-15 describes the fields of the ENLNDAR*n* registers.

**Table 15-15. ENLNDAR*n* Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 0–27 | — | Reserved |
| 28–31 | ENLNDA | Next link descriptor extended address bits (upper 4 bits of 36-bit address) |

### 15.3.1.10 Current List Descriptor Address Registers (CLSDAR*n* and ECLSDAR*n*)

The current list descriptor address registers, shown in Figure 15-17 and Table 15-17, contain the current address of the list descriptor in memory in extended chaining mode.

In extended chaining mode, software must initialize CLSDAR*n* and ECLSDAR*n* to point to the first list descriptor in memory. After finishing the last link descriptor in the current list, the DMA controller loads the contents of the next list descriptor address register into the current list descriptor address register. If NLSDAR*n*[EOLSD] in the next list descriptor address register is clear, the DMA controller reads the new current list descriptor from memory to process that list. If EOLSD in the next list descriptor address register is set and the last link in the current list is finished all DMA transfers are complete.

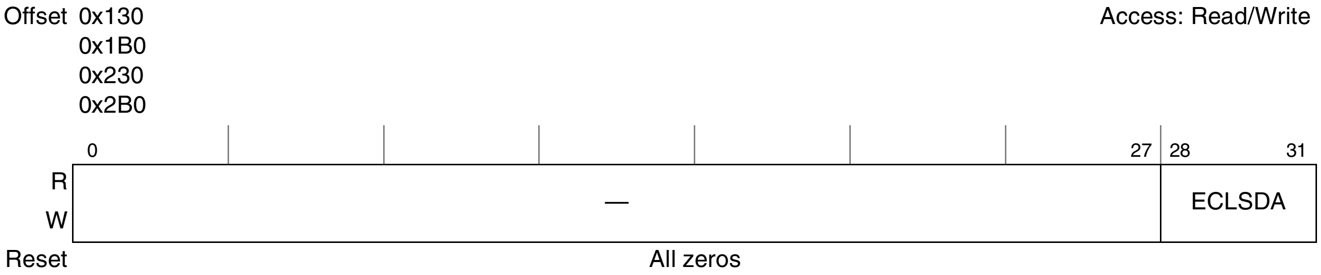


**Figure 15-16. Extended Current List Descriptor Address Registers (ECLSDAR*n*)**

Table 15-16 describes the fields of the ECLSDAR*n* registers.

**Table 15-16. ECLSDAR*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–27 | — | Reserved |
| 28–31 | ECLSDA | Current list descriptor extended address bits (upper 4 bits of 36-bit address) |

Figure 15-17 describes the definition for the CLSDAR*n* registers.

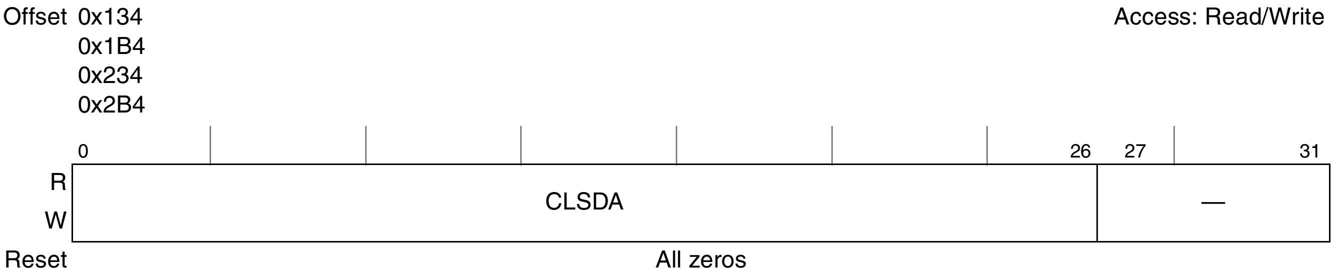


**Figure 15-17. Current List Descriptor Address Registers (CLSDAR*n*)**

Table 15-17 describes the fields of the CLSDAR*n*.

**Table 15-17. CLSDAR*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–26 | CLSDA | Current list descriptor address. Contains the low-order bits of the 36-bit current list descriptor address of the buffer descriptor in memory in extended chaining mode. The descriptor must be aligned to a 32-byte boundary. |
| 27–31 | — | Reserved |

### 15.3.1.11 Next List Descriptor Address Registers (NLSDAR*n* and ENLSDAR*n*)

The next list descriptor address registers, shown in Figure 15-18 and Figure 15-19, contain the address for the next list descriptor in memory. If the contents are transferred to the current list descriptor address register they become effective for the current transfer in extended chaining mode.

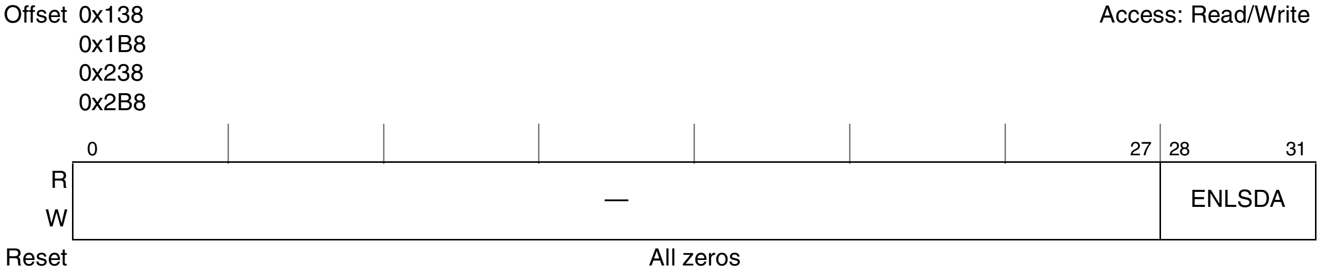


**Figure 15-18. Extended Next List Descriptor Address Registers (ENLSDAR*n*)**

Table 15-17 describes the fields of the ENLSDAR*n*.

**Table 15-18. ENLSDAR*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–27 | — | Reserved |
| 28–31 | ENLSDA | Next list descriptor extended address bits (upper 4 bits of 36-bit address) |

Figure 15-19 describes the definition for the NLSDAR*n* registers.

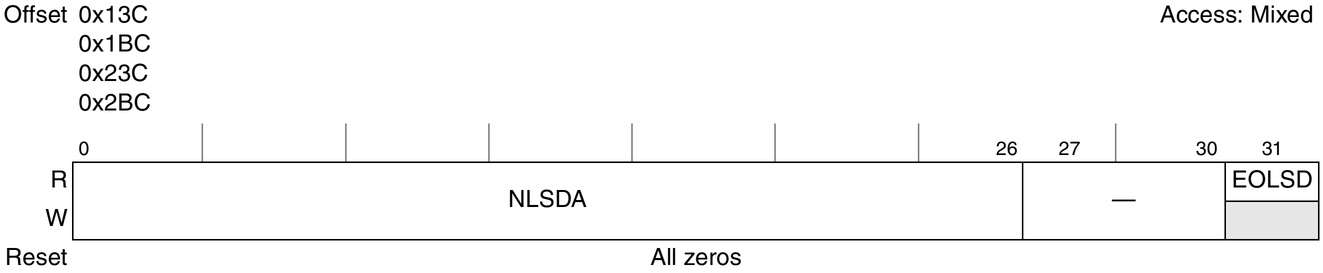


**Figure 15-19. Next List Descriptor Address Registers (NLSDAR*n*)**

Table 15-19 describes the fields of the NLSDAR*n*.

**Table 15-19. NLSDAR*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–26 | NLSDA | Next list descriptor address. Contains the low-order bits of the 36-bit next descriptor address of the buffer descriptor in memory. The descriptor must be aligned on a 32-byte boundary. |
| 27–30 | — | Reserved |
| 31 | EOLSD | End-of-lists descriptor. This bit is ignored in direct mode.<br>0 This list descriptor is not the last list descriptor in memory.<br>1 This list descriptor is the last list descriptor in memory. If this bit is set, then the DMA controller halts after the last link descriptor transaction is finished. |

### 15.3.1.12 Source Stride Registers (SSR*n*)

The source stride register, shown in Figure 15-20, contains the stride size and distance. Note that the source stride information is loaded when a new list descriptor is read from memory. Therefore, the source stride register is applicable for all link descriptors in the new list. Changing the source stride information for a link requires that a new list be generated.

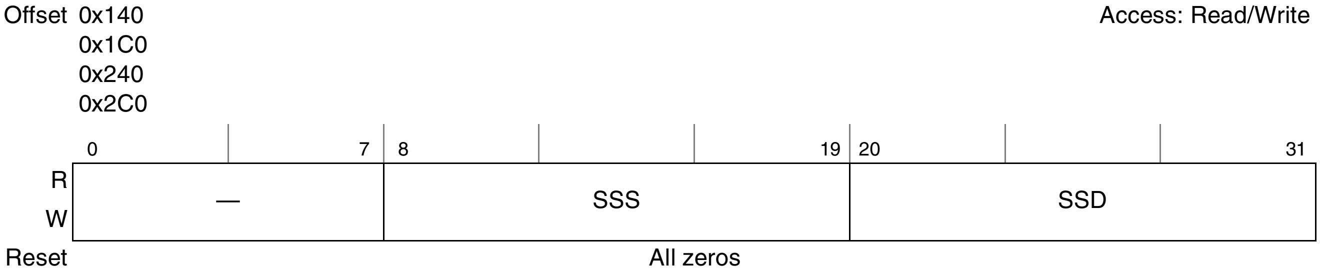


**Figure 15-20. Source Stride Registers (SSR*n*)**

Table 15-20 describes the fields of the SSR*n*.

**Table 15-20. SSR*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–7 | — | Reserved |
| 8–19 | SSS | Source stride size. Number of bytes to transfer before jumping to the next address as specified in the source stride distance field. |
| 20–31 | SSD | Source stride distance. The source stride distance in bytes from start byte to start byte. |

### 15.3.1.13 Destination Stride Registers (DSR*n*)

The destination stride register contains the stride size, and distance. Note that the destination stride information is loaded when a new list descriptor is read from memory. Therefore, the destination stride register is applicable for all link descriptors in the new list. Changing the destination stride information for a link requires that a new list be generated. Figure 15-21 describes the DSR*n*.

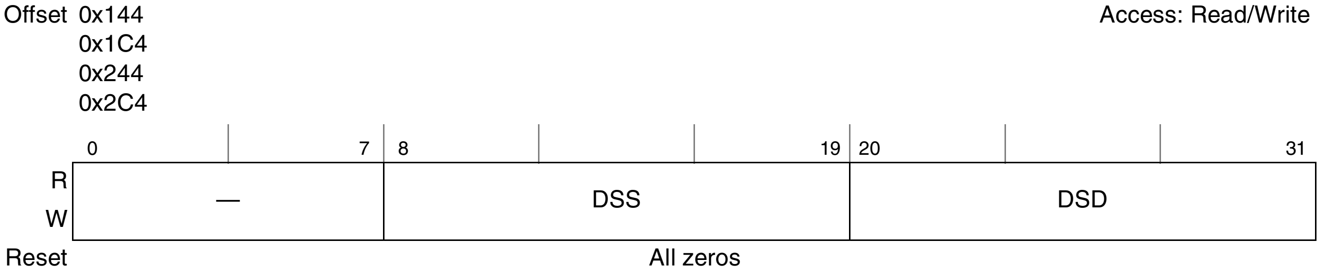


**Figure 15-21. Destination Stride Registers (DSR*n*)**

Table 15-21 describes the fields of the DSR*n*.

**Table 15-21. DSR*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–7 | — | Reserved |
| 8–19 | DSS | Destination stride size. Number of bytes to transfer before jumping to the next address as specified in the destination stride distance field. |
| 20–31 | DSD | Destination stride distance. The destination stride distance in bytes from start byte to start byte. |

### 15.3.1.14 DMA General Status Register (DGSR)

The DMA general status register combines all of the status bits from each channel into one register. This register is read-only. Figure 15-22 describes the DGSR.

Offset 0x300 Access: Read only

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | TE0 | — | CH0 | PE0 | EOLNI0 | CB0 | EOSI0 | EOLSI0 | TE1 | — | CH1 | PE1 | EOLNI1 | CB1 | EOSI1 | EOLSI1 |
| W | | | | | | | | | | | | | | | | |
| Reset | All zeros | | | | | | | | | | | | | | | |

| | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | TE2 | — | CH2 | PE2 | EOLNI2 | CB2 | EOSI2 | EOLSI2 | TE3 | — | CH3 | PE3 | EOLNI3 | CB3 | EOSI3 | EOLSI3 |
| W | | | | | | | | | | | | | | | | |
| Reset | All zeros | | | | | | | | | | | | | | | |

**Figure 15-22. DMA General Status Register (DGSR)**

Table 15-22 describes the fields of the DGSR.

**Table 15-22. DGSR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | TE0 | Transfer error, channel 0<br>0 Normal operation<br>1 An error condition occurred during the DMA transfer. |
| 1 | — | Reserved |
| 2 | CH0 | Channel halted, channel 0 |
| 3 | PE0 | Programming error, channel 0 |
| 4 | EOLNI0 | End-of-links interrupt, channel 0 |
| 5 | CB0 | Channel busy, channel 0 |
| 6 | EOSI0 | End-of-segment interrupt, channel 0 |
| 7 | EOLSI0 | End-of-lists/direct interrupt, channel 0 |
| 8 | TE1 | Transfer error, channel 1<br>0 Normal operation<br>1 An error condition occurred during the DMA transfer. |
| 9 | — | Reserved |

Table 15-22. DGSR Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 10 | CH1 | Channel halted, channel 1 |
| 11 | PE1 | Programming error, channel 1 |
| 12 | EOLNI1 | End-of-links interrupt, channel 1 |
| 13 | CB1 | Channel busy, channel 1 |
| 14 | EOSI1 | End-of-segment interrupt, channel 1 |
| 15 | EOLSI1 | End-of-lists/direct interrupt, channel 1 |
| 16 | TE2 | Transfer error, channel 2<br>0 Normal operation<br>1 An error condition occurred during the DMA transfer. |
| 17 | — | Reserved |
| 18 | CH2 | Channel halted, channel 2 |
| 19 | PE2 | Programming error, channel 2 |
| 20 | EOLNI2 | End-of-links interrupt, channel 2 |
| 21 | CB2 | Channel busy, channel 2 |
| 22 | EOSI2 | End-of-segment interrupt, channel 2 |
| 23 | EOLSI2 | End-of-lists/direct interrupt, channel 2 |
| 24 | TE3 | Transfer error, channel 3<br>0 Normal operation<br>1 An error condition occurred during the DMA transfer. |
| 25 | — | Reserved |
| 26 | CH3 | Channel halted, channel 3 |
| 27 | PE3 | Programming error, channel 3 |
| 28 | EOLNI3 | End-of-links interrupt, channel 3 |
| 29 | CB3 | Channel busy, channel 3 |
| 30 | EOSI3 | End-of-segment interrupt, channel 3 |
| 31 | EOLSI3 | End-of-lists/direct interrupt, channel 3 |

## 15.4 Functional Description

This section describes the function of the DMA controller.

### 15.4.1 DMA Channel Operation

All DMA channels support two different modes of operation: a basic mode (MR*n*[XFE] is cleared) and an extended mode (MR*n*[XFE] is set). In both modes, a channel can be activated by clearing and setting MR*n*[CS], or through the single-write start mode using MR*n*[CDSM/SWSM] and MR*n*[SRW], or through an external control mode using MR*n*[ECS_EN].

In basic mode, the channel can be programmed in basic direct mode or basic chaining mode. In extended mode, the channel can be programmed in extended direct mode or extended chaining mode. Extended mode provides more capabilities, such as extended descriptor chaining, striding capabilities, and a more flexible descriptor structure.

The DMA controller supports misaligned transfers for both the source and destination addresses. In order to maximize performance, the source and destination engines align the source and destination addresses to a 64-byte boundary. The DMA always reads/writes the maximum number of bytes for a given transfer as described by the capability inputs of the DMA controller except for globally coherent transactions that use the size of the cache coherence granule as described by the mode select input.

The DMA controller supports bandwidth control, which prevents a channel from consuming all the data bandwidth in the controller. Each channel is allowed to consume the bandwidth of the shared resources as specified by the bandwidth control value. After the channel uses its allotted bandwidth, the arbiter grants the next channel access to the shared resources. The arbitration is round robin between the channels. This feature is also used to implement the external control pause feature. If the external control start and pause are enabled in the MR*n*, the channel enters a paused state after transferring the data described in the bandwidth control. External control can restart the channel from a paused state.

The DMA programming model permits software to program each DMA engine independently to interrupt on completed segment, chain, or error. It also provides the capability for software to resume the DMA engine from a hardware halted condition by setting the channel continue bit, MR*n*[CC]. See Table 15-23 for more complete descriptions of the channel states and state transitions.

### 15.4.1.1 Basic DMA Mode Transfer

This mode is primarily included for backward compatibility with existing DMA controllers which use a simple programming model. This is the default mode out of reset. The different modes of operation under the basic mode are explained in the following sections.

#### 15.4.1.1.1 Basic Direct Mode

In basic direct mode, the DMA controller does not read descriptors from memory, but instead uses the current parameters programmed in the DMA registers to start the DMA transfer. Software is responsible for initializing SAR*n*, SATR*n*, DAR*n*, DATR*n*, and BCR*n* registers. The DMA transfer is started when MR*n*[CS] is set. Software is expected to program all the appropriate registers before setting MR*n*[CS] to a 1. The transfer is finished after all the bytes specified in the byte count register have been transferred or if an error condition occurs. The sequence of events to start and complete a transfer in basic direct mode is as follows:

1. Poll the channel state (see Table 15-23), to confirm that the specific DMA channel is idle.
2. Initialize SAR*n*, SATR*n*, DAR*n*, DATR*n* and BCR*n*.
3. Set the mode register channel transfer mode bit, MR*n*[CTM], to indicate direct mode. Other control parameters may also be initialized in the mode register.
4. Clear, then set the mode register channel start bit, MR*n*[CS], to start the DMA transfer.
5. SR*n*[CB] is set by the DMA controller to indicate the DMA transfer is in progress.

6. SR*n*[CB] is automatically cleared by the DMA controller after the transfer is finished, or if the transfer is aborted (MR*n*[CA] transitions from a 0 to 1), or if a transfer error occurs.
7. End of segment interrupt is generated if MR*n*[EOSIE] is set.

#### 15.4.1.1.2 Basic Direct Single-Write Start Mode

In basic direct single-write start mode, the DMA controller does not read descriptors from memory, but instead uses the current parameters programmed in the DMA registers to start the DMA transfer. Software is responsible for initializing the SATR*n*, DATR*n*, and BCR*n* registers. Setting MR*n*[SRW] configures the DMA controller to begin the DMA transfer either when SAR*n* is written or when DAR*n* is written, determined by the state of MR*n*[CDSM/SWSM]. Writing to SAR*n* initiates the DMA transfer if MR*n*[CDSM/SWSM] is set. Writing to DAR*n* initiates the DMA transfer if MR*n*[CDSM/SWSM] is cleared. The DMA controller automatically sets the channel start bit, MR*n*[CS]. Software is expected to program all the appropriate registers before writing the source or destination address registers. The transfer is finished after all the bytes specified in the byte count register have been transferred or if an error condition occurs. The sequence of events to start and complete a transfer in single-write start basic direct mode is as follows:

1. Poll the channel state (see Table 15-23), to confirm that the specific DMA channel is idle.
2. Initialize the source attributes (SATR*n*), DATR*n*, and BCR*n* registers.
3. Set the mode register channel transfer mode bit, MR*n*[CTM], and the single-write start direct mode bit, MR*n*[SRW]. Other control parameters may also be initialized in the mode register. Set MR*n*[CDSM/SWSM] for transfers started using SAR*n*. Clear MR*n*[CDSM/SWSM] for transfers started using the DAR*n*.
4. A write to the source or destination address register starts the DMA transfer and automatically sets MR*n*[CS].
5. SR*n*[CB] is set by the DMA controller to indicate the DMA transfer is in progress.
6. SR*n*[CB] is automatically cleared by the DMA controller after the transfer is finished, or if the transfer is aborted (MR*n*[CA] transitions from a 0 to 1), or if a transfer error occurs.
7. End of segment interrupt is generated if MR*n*[EOSIE] is set.

#### 15.4.1.1.3 Basic Chaining Mode

In basic chaining mode, software must first build link descriptor segments in memory. Then the current link descriptor address register must be initialized to point to the first descriptor in memory. The DMA controller loads descriptors from memory prior to a DMA transfer. The DMA controller begins the transfer according to the link descriptor information loaded for the segment. After the current segment is finished, the DMA controller reads the next link descriptor from memory and begins another DMA transfer. The transfer is finished if the current link descriptor is the last one in memory or if an error condition occurs. The sequence of events to start and complete a transfer in chaining mode is as follows:

1. Build link descriptor segments in memory.
2. Poll the channel state (see Table 15-23), to confirm that the specific DMA channel is idle.
3. Initialize CLNDAR*n* and ECLNDAR*n* to point to the first link descriptor in memory.

4. Clear the mode register channel transfer mode bit, MR*n*[CTM], as well as MR*n*[XFE], to indicate basic chaining mode. Other control parameters may also be initialized in the mode register.
5. Clear, then set the mode register channel start bit, MR*n*[CS], to start the DMA transfer.
6. SR*n*[CB] is set by the DMA controller to indicate the DMA transfer is in progress.
7. SR*n*[CB] is automatically cleared by the DMA controller after finishing the transfer of the last descriptor segment, or if the transfer is aborted (MR*n*[CA] transitions from a 0 to 1), or if an error occurs during any of the transfers.

#### 15.4.1.1.4 Basic Chaining Single-Write Start Mode

Basic chaining single-write start mode allows a chain to be started by writing the current link descriptor address register (CLNDAR*n*). (Note that ECLNDAR*n* must be written *first* so that the full 36-bit descriptor address is present when the chain starts.) Setting MR*n*[CDSM/SWSM] in the mode register causes MR*n*[CS] to be automatically set when the current link descriptor address register is written. The sequence of events to start and complete a chain using single-write start mode is as follows:

1. Set the mode register current descriptor start mode bit, MR*n*[CDSM/SWSM], and the extended features enable bit MR*n*[XFE]. Also, clear the channel transfer mode bit, MR*n*[CTM]. This initialization indicates basic chaining and single-write start mode. Also other control parameters may be initialized in the mode register.
2. Build link descriptor segments in memory.
3. Poll the channel state (see Table 15-23), to confirm that the specific DMA channel is idle.
4. Initialize CLNDAR*n* and ECLNDAR*n* to point to the first descriptor segment in memory. This write automatically causes the DMA controller to begin the link descriptor fetch and set MR*n*[CS].
5. SR*n*[CB] is set by the DMA controller to indicate the DMA transfer is in progress.
6. SR*n*[CB] is automatically cleared by the DMA controller after finishing the transfer of the last descriptor segment, or if the transfer is aborted (MR*n*[CA] transitions from a 0 to 1), or if an error occurs during any of the transfers.

### 15.4.1.2 Extended DMA Mode Transfer

The extended DMA mode also operates in chaining and direct mode. It offers additional capability over the basic mode by supporting striding and a more flexible descriptor structure. This additional functionality also requires a new and more complex programming model. The extended DMA mode is activated by setting MR*n*[XFE].

#### 15.4.1.2.1 Extended Direct Mode

Extended direct mode has the same functionality as basic direct mode with the addition of stride capabilities. The bit settings are the same as in direct mode with the exception of the MR*n*[XFE] being set. Striding on the source address can be accomplished by setting SATR*n*[SSME] and setting the desired stride size and distance in SSR*n*. Striding on the destination address can be accomplished by setting DATR*n*[DSME] and setting the desired stride size and distance in DSR*n*.

#### 15.4.1.2.2 Extended Direct Single-Write Start Mode

Extended direct single-write start mode has the same functionality as the basic direct single-write start mode with the addition of stride capabilities. The bit settings are also the same with the exception of MR*n*[XFE] being set. Striding on the source address can be accomplished by setting SATR*n*[SSME] and setting the desired stride size and distance in SSR*n*. Striding on the destination address can be accomplished by setting DATR*n*[DSME] and setting the desired stride size and distance in DSR*n*.

#### 15.4.1.2.3 Extended Chaining Mode

In extended chaining mode, the software must first build list and link descriptor segments in memory. Then CLSDAR*n* and ECLSDAR*n* must be initialized to point to the first list descriptor in memory. The DMA controller loads list descriptors and link descriptors from memory prior to a DMA transfer. The DMA controller begins the transfer according to the link descriptor information loaded. Once the current link descriptor is finished, the DMA controller reads the next link descriptor from memory and begins another DMA transfer. If the current link descriptor is the last in the list, the DMA controller reads the next list descriptor in memory. The transfer is finished if the current link descriptor is the last one in the last list in memory or if an error condition occurs. The sequence of events to start and complete a transfer in extended chaining mode is as follows:

1. Build link and list descriptor segments in memory.
2. Poll the channel state (see Table 15-23), to confirm that the specific DMA channel is idle.
3. Initialize CLSDAR*n* and ECLSDAR*n* to point to the first list descriptor in memory.
4. Clear the mode register channel transfer mode bit, MR*n*[CTM], to indicate chaining mode. MR*n*[XFE] must be set to indicate extended DMA mode. Other control parameters may also be initialized in the mode register.
5. Clear, then set the mode register channel start bit, MR*n*[CS], to start the DMA transfer.
6. SR*n*[CB] is set by the DMA controller to indicate the DMA transfer is in progress.
7. SR*n*[CB] is automatically cleared by the DMA controller after finishing the transfer of the last descriptor segment, or if the transfer is aborted (MR*n*[CA] transitions from a 0 to 1), or if an error occurs during any of the transfers.

#### 15.4.1.2.4 Extended Chaining Single-Write Start Mode

In the extended mode, the single-write start feature allows a chain to be started by writing the current list descriptor pointer. Setting MR*n*[CDSM/SWSM] causes MR*n*[CS] to be set automatically when CLSDAR*n* is written. (Note that ECLSDAR*n* must be written *first* so that the full 36-bit descriptor address is present when the chain starts.) The sequence of events to start and complete an extended chain using single-write start mode is as follows:

1. Set MR*n*[CDSM/SWSM], MR*n*[CTM], and MR*n*[XFE] to indicate extended chaining and single-write start mode. Also other control parameters may be initialized in the mode register.
2. Build list and link descriptor segments in local memory.
3. Poll the channel state (see Table 15-23), to confirm that the specific DMA channel is idle.

4. Initialize the current list descriptor address register to point to the first list descriptor segment in memory. This write automatically causes the DMA controller to begin the list descriptor fetch and set MR*n*[CS].
5. SR*n*[CB] is set by the DMA controller to indicate the DMA transfer is in progress.
6. SR*n*[CB] is automatically cleared by the DMA controller after finishing the transfer of the last descriptor segment, or if the transfer is aborted (MR*n*[CA] transitions from a 0 to 1), or if an error occurs during any of the transfers.

### 15.4.1.3 External Control Mode Transfer

An external control can be used to control all DMA channels by setting MR*n*[EMS_EN]. The external control can direct the DMA channel in the following transfer modes:

- Basic direct
- Basic chaining
- Extended direct
- Extended chaining

Note that when operating the DMA in chaining mode the register byte count field, BCR[BC], must be initialized to zero before enabling the pause feature. In chaining modes, the channel does not pause for descriptor fetch transfer.

The external control and the DMA controller use a well defined protocol to communicate. The external control can start or restart a paused DMA transfer. The DMA controller acknowledges a DMA transfer in progress and also indicates a transfer completion. Note that external control cannot cause a channel to enter a paused state.

The pause feature can be enabled by setting MR*n*[EMP_EN]. MR*n*[BWC] specifies how much data to allow a specific channel to transfer before entering a paused state by clearing MR*n*[CS]. Note, however, that write data for a paused transfer may not have reached the target interface when so indicated. The channel can be restarted from a paused state by the asserted edge of $\overline{\text{DREQ}}$ as driven by an external master. In chaining modes, the channel does not pause for descriptor fetch transfer; it only pauses during the actual data transfer.

The following signals are defined for the external control interface:

- $\overline{\text{DMA\_DREQ}}$—Asserting edge triggers a DMA transfer start or restart from a pause request. Sets MR*n*[CS]. (Note that negating $\overline{\text{DMA\_DREQ}}$ does NOT clear MR*n*[CS].)
- $\overline{\text{DMA\_DACK}}$—Indicates a DMA transfer currently in progress. SR*n*[CB] is set.
- $\overline{\text{DMA\_DDONE}}$—Indicates the completion of the DMA controller’s involvement in the transfer and the readiness to accept a new DMA command. SR*n*[CB] is clear. Note, however, that write data may still be queued at the target interface or in the process of transfer on an external interface.

Detailed descriptions of the external control interface are in Table 15-3. The timing diagram of the external control interface is shown in Figure 15-23.

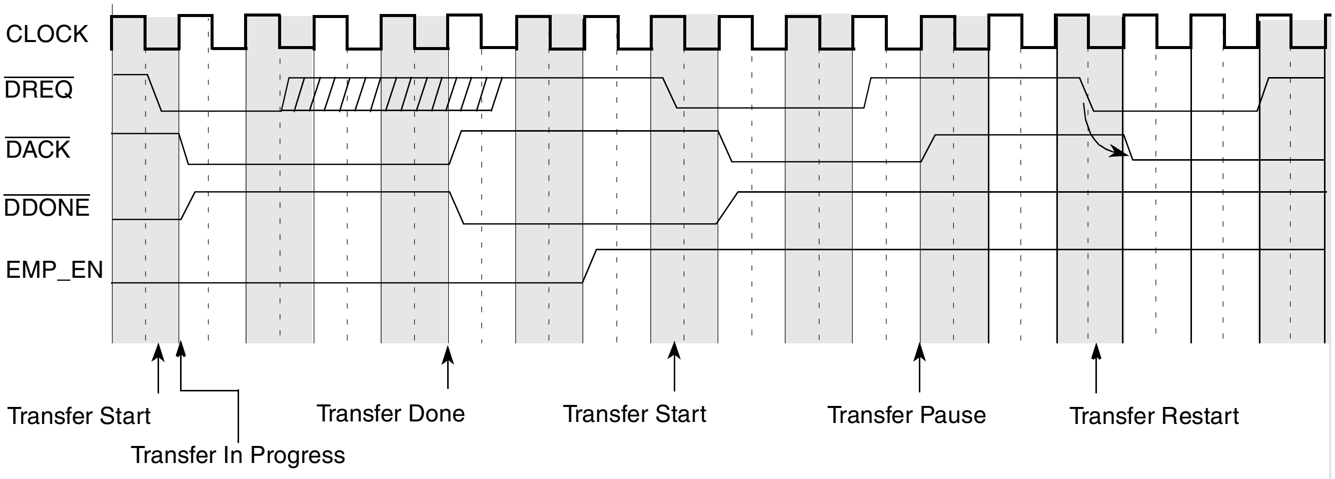


**Figure 15-23. External Control Interface Timing**

### 15.4.1.4 Channel Continue Mode for Cascading Transfer Chains

The channel continue mode (enabled when MR*n*[CC] is set) offers software the flexibility of having the DMA controller get started on descriptors that have already been programmed while software continues to build more descriptors in memory. Software can set the end-of-links descriptor (EOLND) in basic mode, or end-of-lists descriptor (EOLSD) in extended mode, to cause the channel to go into a halted state while software continues to build other descriptors in memory. Software can then set CC to force hardware to continue where it left off. Channel continue is only meaningful for chaining modes, not direct mode.

If CC is set by software while the channel is busy with a transfer, the DMA controller finishes all transfers until it reaches the EOLND in basic mode or EOLSD in extended mode. The DMA controller then refetches the last link descriptor in basic mode, or the last list descriptor in extended mode and clears the channel continue bit. If EOLND or EOLSD is still set for their respective modes, the DMA controller remains in the idle state. If EOLND or EOLSD is not set, the DMA controller continues the transfer by refetching the new descriptor. The channel busy (SR*n*[CB]) bit is cleared when the DMA controller reaches EOLND/EOLSD and is set again when it initiates the refetch of the link or list descriptor.

If CC is set by software while the channel is not busy with a transfer, the DMA controller refetches the last link descriptor in basic mode, or the last list descriptor in extended mode and clears the channel continue bit. If EOLND or EOLSD is still set for their respective modes, the DMA controller remains in the idle state. If the EOLND or EOLSD bits are not set, the DMA controller continues the transfer by refetching the new descriptor.

#### 15.4.1.4.1 Basic Mode

On a channel continue, the descriptor at the current link descriptor address registers (CLNDAR*n* and ECLNDAR*n*) is refetched to get the next link descriptor address field as updated by software. The channel halts if NLNDAR*n*[EOLND] is still set. If EOLND is zero, the next link descriptor address is copied into CLNDAR*n* and ECLNDAR*n* and the channel continues with another descriptor fetch of the current link descriptor address. As a result, two link descriptor fetches always exist after channel continue before starting the first transfer.

#### 15.4.1.4.2 Extended Mode

On a channel continue, the descriptor at the current list descriptor (CLSDAR*n* and ECLSDAR*n*) address register is refetched to get the next list descriptor address field as updated by software. The channel halts if NLSDAR*n*[EOLSD] is still set. If not, the next list descriptor address is copied into the CLSDAR*n* and ECLSDAR*n* registers and the channel continues with another descriptor fetch of the current list descriptor address. As a result, two list descriptor fetches always exist after channel continue before the first link descriptor fetch and the first transfer.

### 15.4.1.5 Channel Abort

Software can abort a previously initiated transfer by setting MR*n*[CA]. Once the DMA channel controller detects a zero-to-one transition of MR*n*[CA], it finishes the current sub-block transfer and halts all further activity. The controller then waits for all previously initiated transfers from the specified channel to drain and clears SR*n*[CB]. Successful completion of a software initiated abort request can be recognized by MR*n*[CA] being set and SR*n*[CB] being cleared. Obviously, if the controller was already halted because of an error condition (SR*n*[TE] is set), or the channel has completed all transfers, then SR*n*[CB] being cleared may not signify that the controller entered a halt state due to the abort request.

### 15.4.1.6 Bandwidth Control

MR*n*[BWC] specifies how much data to allow a specific channel to transfer before allowing the next channel to use the shared data transfer hardware. This promotes equitable bandwidth allocation between channels. However, if only one channel is busy, hardware overrides the specified bandwidth control size value. The DMA controller allows a channel to transfer up to 1 Kbyte at a time when no other channel is active.

### 15.4.1.7 Channel State

Table 15-23 defines the state of a channel based on the values of the channel start (MR*n*[CS]), channel busy (SR*n*[CB]), transfer error (SR*n*[TE]), and channel continue (MR*n*[CC]) bits.

**Table 15-23. Channel State Table**

| MR*n*[CS] | SR*n*[CB] | SR*n*[TE] | MR*n*[CC] | Channel State |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | Idle state. This is the state of the bits out of reset. |
| 0 | 0 | 0 | 1 | Channel continue unexpected. Channel remains idle |
| 0 | 0 | 1 | 0 | Error occurred after software halted the channel. |
| 0 | 0 | 1 | 1 | Channel Continue unexpected. Channel remains in error halt state |
| 0 | 1 | 0 | 0 | Software halted channel. The channel was busy and software cleared MR*n*[CS]. |
| 0 | 1 | 0 | 1 | Channel remains in halt state. |
| — | 1 | 1 | — | The channel has encountered an error condition and it is trying to halt. |
| 1 | 0 | 0 | 0 | Ready to start a transfer, or transfer completed |
| 1 | 0 | 0 | 1 | Continue transfer (only meaningful in chaining mode, not direct mode). In direct mode, the channel continue has no effect. |

Table 15-23. Channel State Table (continued)

| MR*n*[CS] | SR*n*[CB] | SR*n*[TE] | MR*n*[CC] | Channel State |
|---|---|---|---|---|
| 1 | 0 | 1 | 0 | Error occurred during transfer |
| 1 | 0 | 1 | 1 | Channel remains in error halt state |
| 1 | 1 | 0 | 0 | Transfer in progress |
| 1 | 1 | 0 | 1 | Continue after reaching the end of list/link, or the first descriptor fetch after channel continue |

### 15.4.1.8 Illustration of Stride Size and Stride Distance

If operating in stride mode, the stride size defines the amount of data to transfer before jumping to the next quantity of data as specified by the stride distance. The stride distance is added to the current base address to point to the next quantity of data to be transferred. Figure 15-24 illustrates the stride size and distance parameters. As shown, each time the stride distance is added to the base address, the resulting address becomes the new base address. This sequence repeats until the amount of data transferred equals the transfer size.

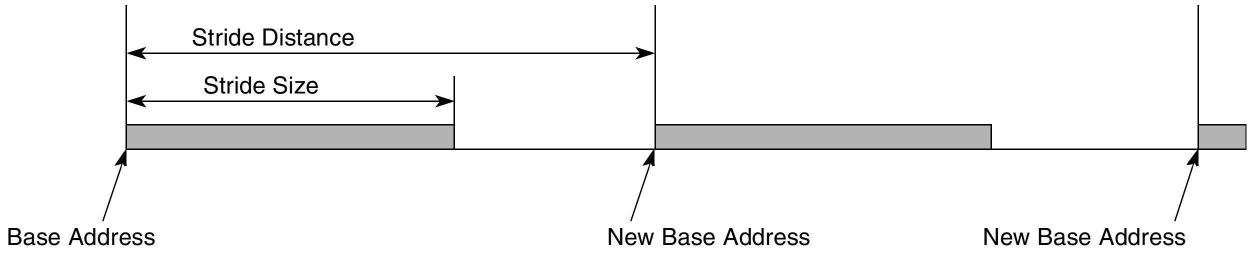


Figure 15-24. Stride Size and Stride Distance

## 15.4.2 DMA Transfer Interfaces

The DMA can be used to achieve data transfers across the entire memory map. Note that a single DMA transfer in any of the direct or chaining modes must not cross a 16GB (34-bit) address boundary.

## 15.4.3 DMA Errors

On a transfer error (uncorrectable ECC errors on memory accesses, parity errors on local bus or PCI, address mapping errors, for example), the DMA halts by setting SR*n*[TE] and generates an interrupt if MR*n*[EIE] is set. On a programming error, the DMA sets SR*n*[PE] and generates an interrupt if MR*n*[EIE] is set. The DMA controller detects the following programming errors:

- Transfer started with a byte count of zero
- Stride transfer started with a stride size of zero
- Transfer started with a priority of three
- Illegal type, defined by SATR*n*[SREADTTYPE] and DATR*n*[DWRITETTYPE], used for the transfer.

### 15.4.4 DMA Descriptors

The DMA engine recognizes list descriptors and link descriptors. List descriptors connect lists of link descriptors. Link descriptors describe the DMA activity that is to take place. DMA descriptors are built in either local or remote memory and are connected by the next descriptor fields. Only link descriptors contain information for the DMA controller to transfer data. Software must ensure that each descriptor is 32-byte aligned. The last link descriptor in the last list in memory sets the NLNDAR*n*[EOLND] bit in the next link descriptor and NLSDAR*n*[EOLSD] in the next list descriptor fields indicating that these are the last descriptors in memory. Software initializes the current list descriptor address register to point to the first list descriptor in memory. The DMA controller traverses through the descriptor lists until the last link descriptor is met. For each link descriptor in the chain, the DMA controller starts a new DMA transfer with the control parameters specified by that descriptor. Link and list descriptor fetches always snoop the local memory space.

**NOTE**

Software must ensure that each descriptor is aligned on a 32-byte boundary.

Table 15-24 summarizes the DMA list descriptors.

**Table 15-24. List DMA Descriptor Summary**

| Descriptor Field | Description |
|---|---|
| Next list descriptor extended address | Points to the next list descriptor in memory. After the DMA controller reads the descriptor from memory, this field is loaded into the next list descriptor extended address registers. |
| Next list descriptor address | Points to the next list descriptor in memory. After the DMA controller reads the descriptor from memory, this field is loaded into the next list descriptor address registers. |
| First link descriptor extended address | Points to the first link descriptor in memory for this list. After the DMA controller reads the descriptor from memory, this field is loaded into the current link descriptor extended address registers. |
| First link descriptor address | Points to the first link descriptor in memory for this list. After the DMA controller reads the descriptor from memory, this field is loaded into the current link descriptor address registers. |
| Source stride | Contains the stride information used for the data source if striding is enabled for a link in the list |
| Destination stride | Contains the stride information used for the data destination if striding is enabled for a link in the list |

Table 15-25 summarizes the DMA link descriptors.

**Table 15-25. Link DMA Descriptor Summary**

| Descriptor Field | Description |
|---|---|
| Source attributes register | Contains source transaction attributes |
| Source address | Contains the source address of the DMA transfer. After the DMA controller reads the descriptor from memory, this field is loaded into the Source address register. |
| Destination attributes register | Contains destination transaction attributes |
| Destination address | Contains the destination address of the DMA transfer. After the DMA controller reads the descriptor from memory, this field is loaded into the destination address register. |
| Next link descriptor extended address | Points to the next link descriptor in memory. After the DMA controller reads the link descriptor from memory, this field is loaded into the extended next link descriptor address registers |

Table 15-25. Link DMA Descriptor Summary (continued)

| Descriptor Field | Description |
|---|---|
| Next link descriptor address | Points to the next link descriptor in memory. After the DMA controller reads the link descriptor from memory, this field is loaded into the next link descriptor address registers. |
| Byte count | Contains the number of bytes to transfer. After the DMA controller reads the descriptor from memory, this field is loaded into the byte count register. |

Figure 15-25 describes the DMA transaction flow.

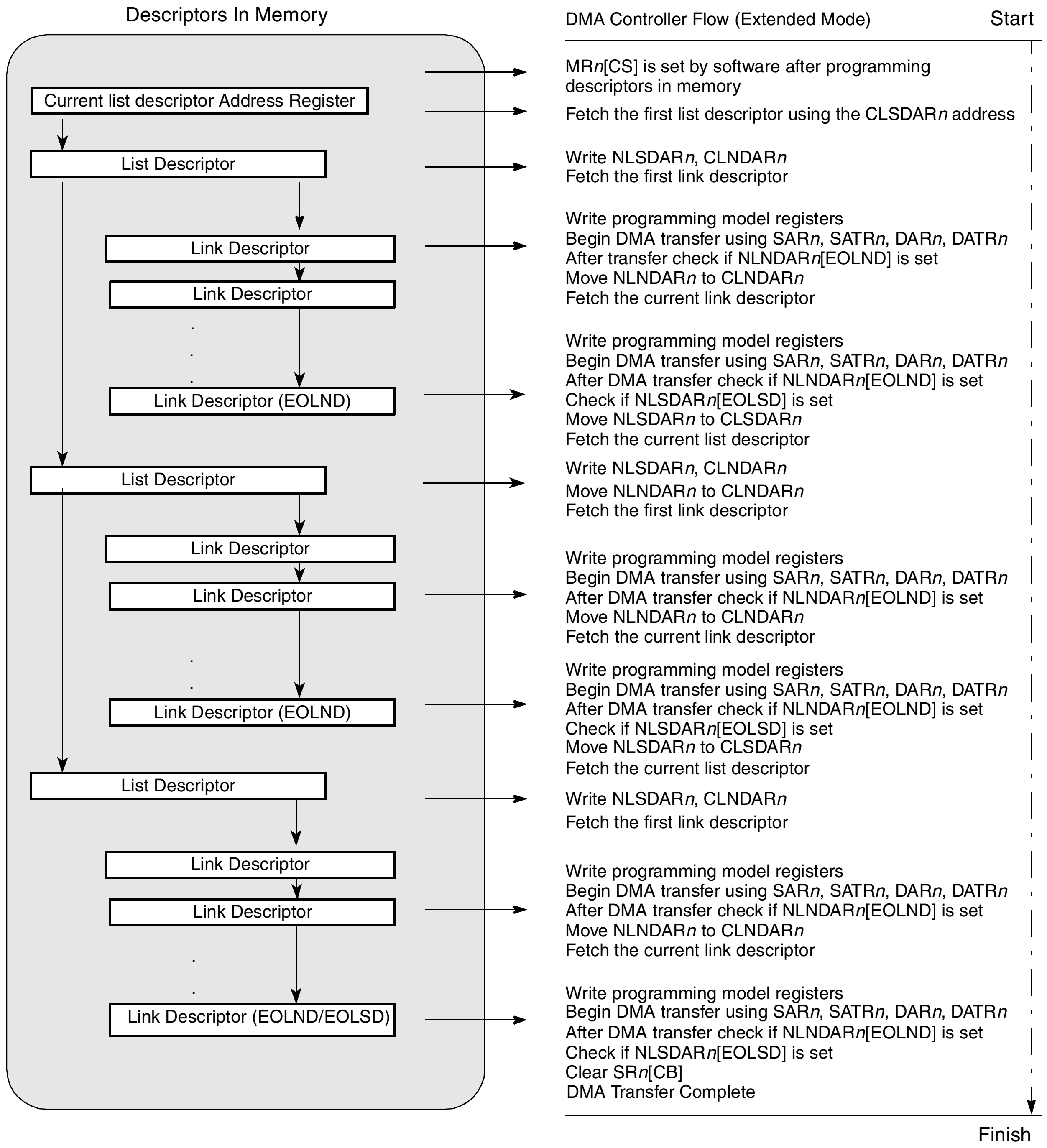

Descriptors In Memory
DMA Controller Flow (Extended Mode)
Start
Current list descriptor Address Register
List Descriptor
Link Descriptor
Link Descriptor
Link Descriptor (EOLND)
List Descriptor
Link Descriptor
Link Descriptor
Link Descriptor (EOLND)
List Descriptor
Link Descriptor
Link Descriptor
Link Descriptor (EOLND/EOLSD)
MRn[CS] is set by software after programming descriptors in memory
Fetch the first list descriptor using the CLSDARn address
Write NLSDARn, CLNDARn
Fetch the first link descriptor
Write programming model registers
Begin DMA transfer using SARn, SATRn, DARn, DATRn
After transfer check if NLNDARn[EOLND] is set
Move NLNDARn to CLNDARn
Fetch the current link descriptor
Write programming model registers
Begin DMA transfer using SARn, SATRn, DARn, DATRn
After DMA transfer check if NLNDARn[EOLND] is set
Check if NLSDARn[EOLSD] is set
Move NLSDARn to CLSDARn
Fetch the current list descriptor
Write NLSDARn, CLNDARn
Move NLNDARn to CLNDARn
Fetch the first link descriptor
Write programming model registers
Begin DMA transfer using SARn, SATRn, DARn, DATRn
After DMA transfer check if NLNDARn[EOLND] is set
Move NLNDARn to CLNDARn
Fetch the current link descriptor
Write programming model registers
Begin DMA transfer using SARn, SATRn, DARn, DATRn
After DMA transfer check if NLNDARn[EOLND] is set
Check if NLSDARn[EOLSD] is set
Move NLSDARn to CLSDARn
Fetch the current list descriptor
Write NLSDARn, CLNDARn
Fetch the first link descriptor
Write programming model registers
Begin DMA transfer using SARn, SATRn, DARn, DATRn
After DMA transfer check if NLNDARn[EOLND] is set
Move NLNDARn to CLNDARn
Fetch the current link descriptor
Write programming model registers
Begin DMA transfer using SARn, SATRn, DARn, DATRn
After DMA transfer check if NLNDARn[EOLND] is set
Check if NLSDARn[EOLSD] is set
Clear SRn[CB]
DMA Transfer Complete
Finish

**Figure 15-25. DMA Transaction Flow with DMA Descriptors**

Figure 15-26 describes the format of the list descriptors.

| Offset | |
|---|---|
| 0x00 | Next List Descriptor Extended Address |
| 0x04 | Next List Descriptor Address |
| 0x08 | First Link Descriptor Extended Address |
| 0x0C | First Link Descriptor Address |
| 0x10 | Source Stride |
| 0x14 | Destination Stride |
| 0x18 | Reserved |
| 0x1C | Reserved |

**Figure 15-26. List Descriptor Format**

Figure 15-27 describes the format of the link descriptors.

| Offset | |
|---|---|
| 0x00 | Source Attributes |
| 0x04 | Source Address |
| 0x08 | Destination Attributes |
| 0x0C | Destination Address |
| 0x10 | Next Link Descriptor Extended Address |
| 0x14 | Next Link Descriptor Address |
| 0x18 | Byte Count |
| 0x1C | Reserved |

**Figure 15-27. Link Descriptor Format**

### 15.4.5 Limitations and Restrictions

This section addresses some of the limitations and restrictions of the DMA controller and is intended to help software maximize the DMA performance and avoid DMA programming errors.

The limitations of the DMA controller are the following:

- Due to the limited number of buffers that the DMA controller can use, stride sizes less than 64 bytes should be avoided. Maximum utilization is obtained from strides greater than or equal to 256 bytes. However, small stride sizes can be used for scatter-gather functions.
- Coherent reads or writes are broken up into cache line accesses in the DMA.

The DMA controller restrictions are as follows:

- All interface capabilities from where descriptors are being fetched must support read sizes of 32 bytes or greater.
- If MR*n*[SAHE] is set, the source interface transfer size capability must be greater than or equal to MR*n*[SAHTS]. The source address must be aligned to a size specified by SAHTS.

- If MR*n*[DAHE] is set, the destination interface transfer size capability must be greater than or equal to MR*n*[DAHTS]. The destination address must be aligned to the size specified by DAHTS.
- Destination striding is not supported if MR*n*[DAHE] is set and source striding is not supported if MR*n*[SAHE] is set.
- A single DMA transfer in any of the direct or chaining modes must not cross a 16GB (34-bit) address boundary

## 15.5 DMA System Considerations

This section provides information about how to make most effective use of the DMA channels.

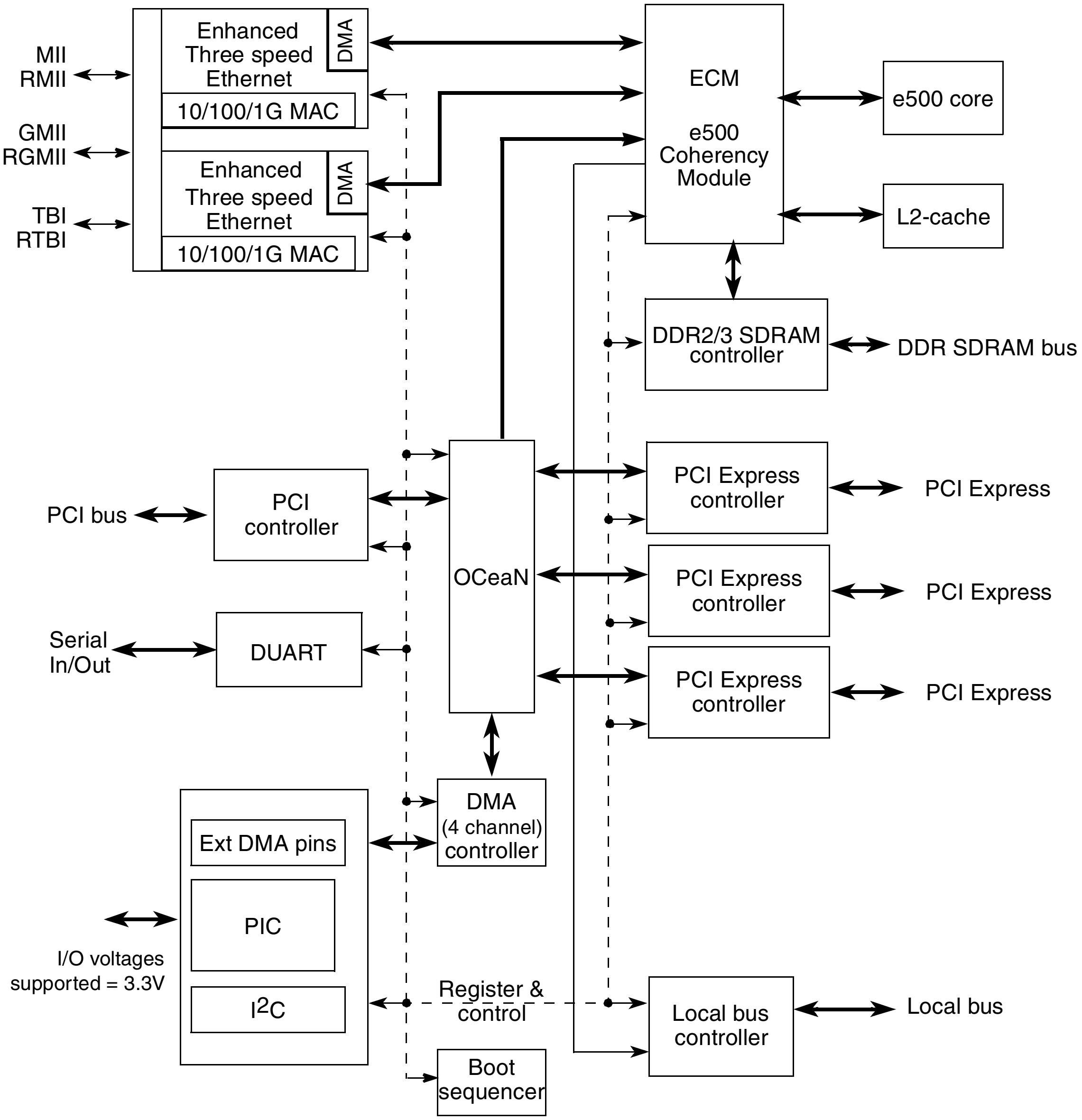


**Figure 15-28. DMA Data Paths**

**Note:** On-chip target configuration registers include I$^2$C data register.

**Note:** On-chip  Ethernet  captive resource. Not available to external masters.

**Note:** On-chip 4-channel controller can serve external masters.

### 15.5.1 Unusual DMA Scenarios

The following is a description of unusual DMA paths including explanations of why some functional blocks cannot serve as DMA targets. The following topics are addressed:

- DMA transaction initiators (masters)
- DMA targets, that is, data sources or destinations
- Transparency of the bus controllers to DMA transactions
- What is useful as opposed to what is possible. For example, any register can be addressed through an internal control bus, which means configuration and control registers can be DMA targets.

#### 15.5.1.1 DMA to Core

The L1 cache cannot be a direct DMA target because it cannot be directly addressed by software. However, DMA access into the L1 cache occurs indirectly if a block of memory that is cached in the L1 is specified as the DMA target. This effect is deterministic if the target memory block was locked into the L1 with cache locking instructions.

#### 15.5.1.2 DMA to Configuration, Control, and Status Registers

Because any internal register can be addressed with the four-channel DMA controller, configuration, control, and status registers throughout the device are valid DMA targets. However, the primary purpose of DMA—to reduce processor load by moving large blocks of data— is not served by DMA transfers of configuration data. For example, while it is possible to DMA into the I$^2$C controller or programmable interrupt controller (PIC), doing so is extremely inefficient and is seldom beneficial in normal operation. The overhead of creating DMA descriptors far exceeds any savings in CPU cycles.

#### 15.5.1.3 DMA to I$^2$C

The I$^2$C controller is not transparent to DMA transfers. Observe the caveats listed in Section 15.5.1.2, “DMA to Configuration, Control, and Status Registers,” when accessing any I$^2$C register, including the data register (I2CDR).

#### 15.5.1.4 DMA to DUART

The DUART provides complete and sophisticated DMA support which is described in Chapter 12, “DUART,” specifically, Section 12.4.5, “FIFO Mode.”

# Chapter 16
# PCI Bus Interface

The PCI interface is compatible with the *PCI Local Bus Specification*, Rev. 2.2. It is beyond the scope of this manual to document the intricacies of the PCI bus. This chapter describes the PCI controller (referenced as PCI throughout this chapter) of this device and provides a basic description of PCI bus operations. The specific emphasis is directed at how the integrated processor implements the PCI bus. Designers of systems incorporating PCI devices should refer to the specification for a thorough description of the PCI bus.

**NOTE**

Much of the available PCI literature refers to a 16-bit quantity as a WORD and a 32-bit quantity as a DWORD. Because this is inconsistent with the terminology in this manual, the terms ‘word’ and ‘double word’ are not used in this chapter. Instead, the number of bits or bytes indicates the exact quantity.

## 16.1 Introduction

The PCI controller acts as a bridge between the PCI interface and the OCeaN switch fabric. Figure 16-1 is a high-level block diagram of the PCI controller.

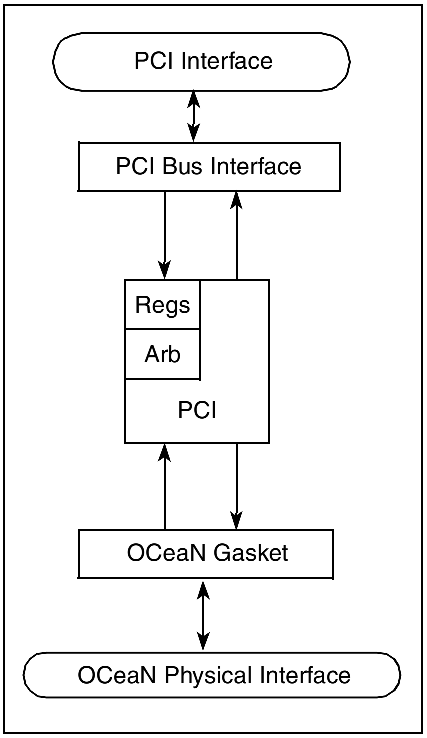


**Figure 16-1. PCI Controller Block Diagram**

## 16.1.1 Overview

The PCI controller connects the OCeaN to the PCI bus, to which I/O components are connected. The PCI bus uses a 32-bit multiplexed address/data bus, plus various control and error signals. The PCI interface supports address and data parity with error checking and reporting.

The integrated processor’s PCI interface functions both as a master (initiator) and a target device. Internally, the design is divided into the following:

- Data path blocks
- Control logic blocks
- Memory

The data path blocks contain the queues, tables for transaction tracking, and ordering. The control blocks contain control logic and state-machines for buffer control, bus protocol, tag generation, and transaction resizing. The memory blocks are used solely for inbound and outbound data storage. This allows the integrated processor to handle separate PCI transactions simultaneously. For example, consider the case where a burst-write transaction from the integrated processor to another PCI device terminates with a disconnect before finishing the transaction. If another PCI device is granted the PCI bus and requests a burst-read from local memory, the integrated processor, as a target, can accept the burst-read transfer. When the integrated processor is granted mastership of the PCI bus, the burst-write transaction continues. The PCI interface does not flush pending outbound writes as a result of an inbound read command. Systems must not rely on inbound reads to ensure all pending outbound writes have completed. For example, consider the case where a core writes data to a PCI device and then updates a flag in the local DDR memory indicating the write to PCI has completed. An external PCI master may misread the flag ahead of the actual write transaction's completion on the PCI bus.

There are two blocks of memory in the design:

- The inbound buffers
- The outbound read buffers combined with the outbound write buffers

There are many blocks of control logic in the block. On the PCI side there are machines for PCI controller initiated address and data tenures for inbound and outbound data, respectively. On the OCeaN side there are machines for fabric arbitration, outbound data, and inbound data.

As an initiator, the integrated processor supports read and write operations to the PCI memory space, the PCI I/O space, and the 256-byte PCI configuration space. As an initiator, the integrated processor also supports generating PCI special-cycle and interrupt-acknowledge transactions. As a target, the integrated processor supports read and write operations to local memory, and, when configured in agent mode, read and write operations to the internal PCI configuration registers.

The integrated processor can function as either a PCI host bridge (host mode) or a peripheral device on the PCI bus (agent mode). See Section 16.1.3.1.1, “Host Mode,” for more information.

In agent mode, all of the PCI configuration registers in the integrated processor can be programmed from the PCI bus. See Section 16.4.2.11.3, “Agent Accessing the PCI Configuration Space,” for more information.

The PCI interface provides bus arbitration for the integrated processor and up to five other PCI bus masters. The arbitration algorithm is a programmable two-level, round-robin priority selector. The on-chip PCI arbiter can operate in both host and agent modes or it can be disabled to allow for an external PCI arbiter.

The integrated processor also provides an address translation mechanism to map inbound PCI to OCeaN accesses and outbound OCeaN to PCI accesses.

#### 16.1.1.1 Outbound Transactions

Upon detecting an OCeaN-to-PCI transaction, the integrated processor requests the use of the PCI bus. For OCeaN-to-PCI bus write operations, the integrated processor requests mastership of the PCI bus when the source completes the write operation to the OCeaN. For OCeaN-to-PCI read operations, the integrated processor requests mastership of the PCI bus when it decodes that the access is for PCI address space.

Once granted, the integrated processor drives the address (PCI_AD[31:0]) and the bus command (PCI_C/$\overline{\text{BE}}$[3:0]) signals.

The master part of the interface can initiate master-abort cycles, recognizes target-abort, target-retry, and target-disconnect cycles, and supports various device selection timings. The master interface does not run fast back-to-back or exclusive accesses.

### 16.1.1.2 Inbound Transactions

Upon detection of a PCI address phase, the integrated processor decodes the address and bus command to determine if the transaction is within the local memory access boundaries. If the transaction is destined for local memory, the target interface latches the address, decodes the PCI bus command, and forwards the transaction to the OCeaN control unit. On writes to local memory, data is forwarded along with the byte enables (if applicable) to the internal control unit. Note that for inbound writes less than 4-bytes, the PCI controller splits the transaction into single byte writes to the target. Thus, the PCI interface cannot be used to perform single beat writes to 16-bit devices on the local bus interface. On reads, the data is driven on the bus and the byte enables (if applicable) determine which byte lanes contain meaningful data.

The target interface of the integrated processor can issue target-abort, target-retry, and target-disconnect cycles. The target interface supports fast back-to-back transactions. The target interface uses the fastest device selection timing.

The integrated processor supports data streaming to and from local memory. This means that data can flow between the processor PCI interface and local memory as long as the internal buffers are not filled.

## 16.1.2 Features

The following is a list of PCI features that is supported:

- PCI interface 2.2 compatible
- 66- and 33-MHz support
- 32-bit PCI interface support on PCI port
- Host and agent mode support
- 64-bit dual address cycle (DAC) support
- On-chip arbitration with support for five high-priority request and grant signal pairs
- Support for accesses to all PCI memory and I/O address spaces
- Support for PCI-to-memory and memory-to-PCI streaming
- Memory prefetching of PCI read accesses
- Support posting of processor-to-PCI and PCI-to-memory writes
- Support selectable snoop for inbound accesses
- PCI configuration registers
- PCI 3.3-V compatible

## 16.1.3 Modes of Operation

A number of parameters that affect the PCI controller modes of operation are determined at power-on reset (POR) by reset configuration signals as described in .” Table 16-1 provides a summary of these modes.

Table 16-1. POR Parameters for PCI Controller

| Parameter | Description | Section/page |
|---|---|---|
| Host/agent configuration | Selects between host and agent mode for the PCI interface. | |
| PCI clocking | Selects between asynchronous or synchronous clocking for PCI | |
| PCI arbiter enable | Enables the on-chip PCI bus arbiter | |
| PCI I/O impedance | Selects the impedance of the PCI I/O drivers | |

### 16.1.3.1 Host/Agent Mode Configuration

The PCI controller can function as either a PCI host bridge (referred to as host mode) or a peripheral device on the PCI bus (referred to as agent mode). Additionally, the PCI controller can operate in agent configuration lock mode. Note that host/agent mode selection is determined at power-up.

#### 16.1.3.1.1 Host Mode

When the device powers up in host mode, all inbound configuration accesses are ignored (and thus master aborted). See Section 16.5.1.1, “Host Mode,” for more information.

#### 16.1.3.1.2 Agent Mode

When the device powers up in agent mode, it acknowledges inbound configuration accesses. See Section 16.5.1.2, “Agent Mode,” for more information. Note that in PCI agent mode, the PCI controller ignores all PCI memory accesses except those to the memory-mapped registers) until inbound address translation is enabled.

#### 16.1.3.1.3 Agent Configuration Lock Mode

When the device powers up in agent configuration lock mode, it retries inbound configuration accesses until the ACL bit in the PCI bus function register is cleared. See Section 16.5.1.3, “Agent Configuration Lock Mode,” for more information.

### 16.1.3.2 PCI Clocking Configuration

The interface can be configured to be clocked asynchronously with a PCI_CLK input or synchronously with the SYSCLK input. The initial value for clocking is determined by a power-on reset configuration signal.

### 16.1.3.3 PCI Arbiter (Internal/External Arbiter) Configuration

The interface can be configured to use an on-chip or off-chip PCI arbiter. The initial value for the arbiter is determined by a power-on reset configuration signal.

### 16.1.3.4 PCI Impedance Configuration

The device has a programmable impedance for PCI bus signals. The initial value for impedance is determined by a power-on reset configuration signal.

## 16.2 External Signal Descriptions

Figure 16-2 shows the external PCI signals.

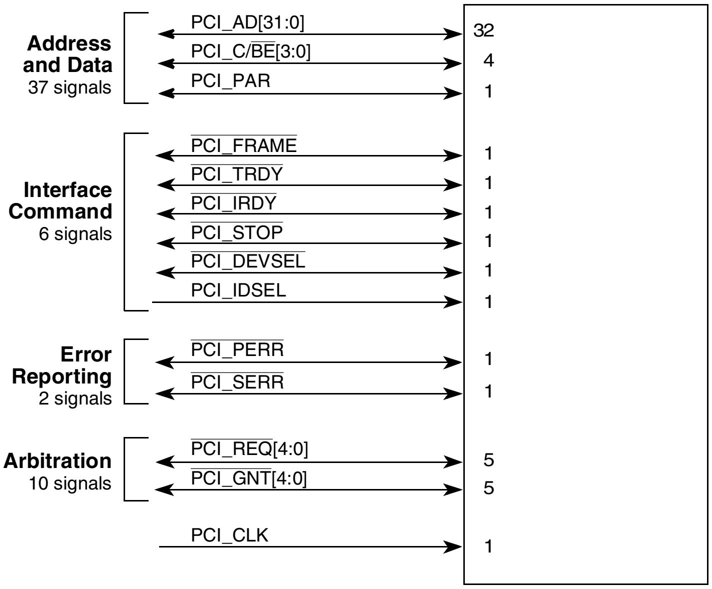


**Figure 16-2. PCI Interface External Signals**

Table 16-2 contains the detailed descriptions of the external PCI interface signals.

**Table 16-2. PCI Interface Signals—Detailed Signal Descriptions**

| Signal | I/O | Description | |
|---|---|---|---|
| PCI_AD[31:0] | I/O | PCI address/data bus. The PCI address/data bus consists of signals that are both input and output signals on this PCI controller. | |
| | O | As outputs for the bidirectional PCI address/data bus, these signals operate as described below. | |
| | | **State Meaning** | Asserted/Negated—Represents the physical address during the address phase of a PCI transaction. During the data phase(s) of a PCI transaction, the PCI address/data bus contain the data being written.<br>The PCI_AD[7:0] signals define the LSB and PCI_AD[31:24] the MSB. |
| | | **Timing** | Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.2 |
| | I | As inputs for the bidirectional PCI address/data bus, these signals operate as described below. | |
| | | **State Meaning** | Asserted/Negated—Represents the address to be decoded as a check for device select during the address phase of a PCI transaction or the data being received during the data phase(s) of a PCI transaction.<br>The PCI_AD[7:0] signals define the LSB and PCI_AD[31:24] the MSB. |
| | | **Timing** | Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.2 |

Table 16-2. PCI Interface Signals—Detailed Signal Descriptions (continued)

| Signal | I/O | Description | |
|---|---|---|---|
| PCI_C/BE[3:0] | I/O | Command/byte enable. The command/byte enable signals are both input and output signals on this PCI controller. The command encodings for PCI bus mode are described in Section 16.4.2.2, "PCI Bus Commands." | |
| | O | As outputs for the bidirectional command/byte enable, these signals operate as described below. | |
| | | **State Meaning** | Asserted/Negated—During the address phase, PCI_C/BE[3:0] define the bus command. During the data phase, PCI_C/BE[3:0] act as byte enables indicating which byte lanes carry meaningful data. The PCI_C/BE[0] signal applies to the LSB. |
| | | **Timing** | Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.2 |
| | I | As inputs for the bidirectional command/byte enable, these signals operate as described below. | |
| | | **State Meaning** | Asserted/Negated—During the address phase, PCI_C/BE[3:0] indicate the command that another master is sending. During the data phase, PCI_C/BE[3:0] indicate which byte lanes are valid. |
| | | **Timing** | Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.2 |
| PCI_DEVSEL | I/O | Device select. The device select signal is both an input and output signal on this PCI controller. | |
| | O | As outputs for the bidirectional device select, these signals operate as described below. | |
| | | **State Meaning** | Asserted—Indicates that this PCI controller has decoded the address and is the target of the current access.<br>Negated—Indicates that this PCI controller has decoded the address and is not the target of the current access. |
| | | **Timing** | Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.2 |
| | I | As inputs for the bidirectional device select, these signals operate as described below. | |
| | | **State Meaning** | Asserted—Indicates that some PCI agent (other than this PCI controller) has decoded its address as the target of the current access.<br>Negated—Indicates that no PCI agent has been selected. |
| | | **Timing** | Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.2 |
| PCI_FRAME | I/O | Frame. The frame signal is both an input and output signal on this PCI controller. | |
| | O | As outputs for the bidirectional frame, these signals operate as described below. | |
| | | **State Meaning** | Asserted—Indicates that this PCI controller, acting as a PCI master, is initiating a bus transaction. While PCI_FRAME is asserted, data transfers may continue.<br>Negated—If PCI_IRDY is asserted, indicates that the PCI transaction is in the final data phase; if PCI_IRDY is negated, indicates that the PCI bus is idle. |
| | | **Timing** | Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.2 |
| | I | As inputs for the bidirectional frame, these signals operate as described below. | |
| | | **State Meaning** | Asserted—Indicates that another PCI master is initiating a bus transaction.<br>Negated—Indicates that the transaction is in the final data phase or that the bus is idle. |
| | | **Timing** | Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.2 |

**Table 16-2. PCI Interface Signals—Detailed Signal Descriptions (continued)**

| Signal | I/O | Description | |
|---|---|---|---|
| PCI_GNT[4:0] | O | PCI bus grant. Output signals on this PCI controller when the arbiter is enabled. When the arbiter is disabled PCI_GNT0 is an input signal. Note that PCI_GNT[*n*] is a point-to-point signal. Every master has its own bus grant signal.<br>**Note:** These signals are also used as reset configuration signals as described in Section 4.4.3, “Power-On Reset Configuration." | |
| | | **State Meaning** | Asserted—Indicates that this PCI controller has granted control of the PCI bus to agent *n*.<br>Negated—Indicates that this PCI controller has not granted control of the PCI bus to agent *n*. |
| | | **Timing** | Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.2 |
| PCI_IDSEL | I | Initialization device select. The initialization device select signal is an input signal on this PCI controller. It is used as a chip select during configuration read and write transactions. | |
| | | **State Meaning** | Asserted—Indicates this PCI controller is being selected as a target of a configuration read or write transactions.<br>Negated—Indicates this PCI controller is not being selected as a target of configuration read or write transactions. |
| | | **Timing** | Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.2 |
| PCI_IRDY | I/O | Initiator ready. The initiator ready signal is both an input and output signal on this PCI controller. | |
| | O | As outputs for the bidirectional initiator ready, these signals operate as described below. | |
| | | **State Meaning** | Asserted—Indicates that this PCI controller, acting as a PCI master, can complete the current data phase of a PCI transaction. During a write, this PCI controller asserts PCI_IRDY to indicate that valid data is present on the data bus. During a read, this PCI controller asserts PCI_IRDY to indicate that it is prepared to accept data.<br>Negated—Indicates that the PCI target needs to wait before this PCI controller, acting as a PCI master, can complete the current data phase. During a write, this PCI controller negates PCI_IRDY to insert a wait cycle when it cannot provide valid data to the target. During a read, this PCI controller negates PCI_IRDY to insert a wait cycle when it cannot accept data from the target. |
| | | **Timing** | Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.2 |
| | I | As inputs for the bidirectional initiator ready, these signals operate as described below. | |
| | | **State Meaning** | Asserted—Indicates another PCI master is able to complete the current data phase of a transaction.<br>Negated—If PCI_FRAME is asserted, indicates a wait cycle from another master. If PCI_FRAME is negated, indicates the PCI bus is idle. |
| | | **Timing** | Assertion/Negation—As specified by PCI Local Bus Specification Rev |

**Table 16-2. PCI Interface Signals—Detailed Signal Descriptions (continued)**

| Signal | I/O | Description | |
|---|---|---|---|
| PCI_PAR | I/O | PCI parity. The PCI parity signal is both an input and output signal on this PCI controller. | |
| | O | As outputs for the bidirectional PCI parity, these signals operate as described below. | |
| | | **State Meaning** | Asserted—Indicates odd parity across the PCI_AD[31:0] and PCI_C/$\overline{\text{BE}}$[3:0] signals during address and data phases.<br>Negated—Indicates even parity across the PCI_AD[31:0] and PCI_C/$\overline{\text{BE}}$[3:0] signals during address and data phases. |
| | | **Timing** | Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.2 |
| | I | As inputs for the bidirectional PCI parity, these signals operate as described below. | |
| | | **State Meaning** | Asserted—Indicates odd parity driven by another PCI master or the PCI target during read data phases.<br>Negated—Indicates even parity driven by another PCI master or the PCI target during read data phases. |
| | | **Timing** | Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.2 |
| $\overline{\text{PCI\_PERR}}$ | I/O | PCI parity error. The PCI parity error signal is both an input and output signal on this PCI controller. | |
| | O | As outputs for the bidirectional PCI parity error, these signals operate as described below. | |
| | | **State Meaning** | Asserted—Indicates that this PCI controller, acting as a PCI agent, detected a data parity error. (The PCI initiator drives $\overline{\text{PCI\_PERR}}$ on read operations; the PCI target drives $\overline{\text{PCI\_PERR}}$ on write operations.)<br>Negated—Indicates no error. |
| | | **Timing** | Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.2 |
| | I | As inputs for the bidirectional PCI parity error, these signals operate as described below. | |
| | | **State Meaning** | Asserted—Indicates that another PCI agent detected a data parity error while this PCI controller was sourcing data (this PCI controller was acting as the PCI initiator during a write, or was acting as the PCI target during a read).<br>Negated—Indicates no error. |
| | | **Timing** | Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.2 |
| $\overline{\text{PCI\_REQ}}$[4:0] | I | PCI bus request. Input signals on this PCI controller when the arbiter is enabled. When the arbiter is disabled, $\overline{\text{PCI\_REQ}}$[0] is an output. Note that PCI_REQ[*n*] is a point-to-point signal. Every master has its own bus request signal. Following is the state meaning for the $\overline{\text{PCI\_REQ}}$[*n*] input. | |
| | | **State Meaning** | Asserted—Indicates that agent *n* is requesting control of the PCI bus to perform a transaction.<br>Negated—Indicates that agent *n* does not require use of the PCI bus. |
| | | **Timing** | Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.2 |

### Table 16-2. PCI Interface Signals—Detailed Signal Descriptions (continued)

| Signal | I/O | Description | |
|---|---|---|---|
| $\overline{\text{PCI\_SERR}}$ | I/O | PCI system error.The PCI system error signal is both an input and output signal on this PCI controller. | |
| | O | As outputs for the bidirectional PCI system error, these signals operate as described below. | |
| | | **State Meaning** | Asserted—Indicates that an address parity error, a target-abort (when this PCI controller is acting as the initiator), or some other system error (where the result is a catastrophic error) was detected.<br>Negated—Indicates no error. |
| | | **Timing** | Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.2 |
| | I | As inputs for the bidirectional PCI system error, these signals operate as described below. | |
| | | **State Meaning** | Asserted—Indicates that a target (other than this PCI controller) has detected a catastrophic error.<br>Negated—Indicates no error. |
| | | **Timing** | Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.2 |
| $\overline{\text{PCI\_STOP}}$ | I/O | Stop.The stop signal is both an input and output signal on this PCI controller. | |
| | O | As outputs for the bidirectional stop, these signals operate as described below. | |
| | | **State Meaning** | Asserted—Indicates that this PCI controller, acting as a PCI target, is requesting that the initiator stop the current transaction.<br>Negated—Indicates that the current transaction can continue. |
| | | **Timing** | Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.2 |
| | I | As inputs for the bidirectional stop, these signals operate as described below. | |
| | | **State Meaning** | Asserted—Indicates that a target is requesting that the PCI initiator stop the current transaction.<br>Negated—Indicates that the current transaction can continue. |
| | | **Timing** | Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.2 |
| $\overline{\text{PCI\_TRDY}}$ | I/O | Target ready. Both an input and output signal on this PCI controller. | |
| | O | As outputs for the bidirectional target ready, these signals operate as described below. | |
| | | **State Meaning** | Asserted—Indicates that this PCI controller, acting as a PCI target, can complete the current data phase of a PCI transaction. During a read, this PCI controller asserts $\overline{\text{PCI\_TRDY}}$ to indicate that valid data is present on the data bus. During a write, this PCI controller asserts $\overline{\text{PCI\_TRDY}}$ to indicate that it is prepared to accept data.<br>Negated—Indicates that the PCI initiator needs to wait before this PCI controller, acting as a PCI target, can complete the current data phase. During a read, this PCI controller negates $\overline{\text{PCI\_TRDY}}$ to insert a wait cycle when it cannot provide valid data to the initiator. During a write, this PCI controller negates $\overline{\text{PCI\_TRDY}}$ to insert a wait cycle when it cannot accept data from the initiator. |
| | | **Timing** | Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.2 |
| | I | As inputs for the bidirectional target ready, these signals operate as described below. | |
| | | **State Meaning** | Asserted—Another PCI target is able to complete the current data phase of a transaction.<br>Negated—Indicates a wait cycle from another target. |
| | | **Timing** | Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.2 |

Table 16-2. PCI Interface Signals—Detailed Signal Descriptions (continued)

| Signal | I/O | Description | |
|---|---|---|---|
| PCI_CLK | I | PCI clock is an independent clock that may be used for the PCI interface. If used the PCI operation is asynchronous with respect to SYSCLK and the platform clock. In order to used this signal as the PCI clock source, it must be designated during POR configuration. See the reset chapter for POR details regarding clock selection as well as proper PCI frequency selection. | |
| | | **Timing** | Assertion/Negation—See the device *Hardware Specification* for specific timing information. |

## 16.3 Memory Map/Register Definitions

The PCI controller supports the following two types of registers:

- Memory-mapped registers—these registers control PCI address translation, PCI error management, and PCI configuration register access. These registers are described in Section 16.3.1, “PCI Memory-Mapped Registers,” and its subsections.
- PCI configuration registers contained within the PCI configuration header—these registers are specified by the PCI bus specification for every PCI device. These registers are described in Section 16.3.2, “PCI Configuration Header,” and its subsections.

### 16.3.1 PCI Memory-Mapped Registers

The PCI memory mapped registers are accessed by reading and writing to an address comprised of the base address (specified in the CCSRBAR on the local side or the PCSRBAR on the PCI side) plus the block base address, plus the offset of the specific register to be accessed. Note that all memory-mapped registers (except the PCI configuration data register, PCI CFG_DATA) must only be accessed as 32-bit quantities.

Table 16-3 lists the memory-mapped registers.

Table 16-3. PCI Memory-Mapped Register Map

| Offset | Register | Access | Reset | Section/page |
|---|---|---|---|---|
| **PCI Controller Memory-Mapped Registers—Block Base Address 0x0_8000** | | | | |
| **PCI Configuration Access Registers** | | | | |
| 0x000 | CFG_ADDR—PCI configuration address | R/W | 0x0000_0000 | 16.3.1.1.1/16-14 |
| 0x004 | CFG_DATA—PCI configuration data | R/W | 0x0000_0000 | 16.3.1.1.2/16-15 |
| 0x008 | INT_ACK—PCI interrupt acknowledge | R | 0x0000_0000 | 16.3.1.1.3/16-15 |
| 0x00C–0xBFC | Reserved | — | — | — |
| **PCI ATMU Registers—Outbound and Inbound** | | | | |
| 0xC00–0xC3C–Outbound Window 0 (default) | | | | |
| 0xC00 | POTAR0—PCI outbound window 0 (default) translation address register | R/W | 0x0000_0000 | 16.3.1.2.1/16-16 |
| 0xC04 | POTEAR0—PCI outbound window 0 (default) translation extended address register | R/W | 0x0000_0000 | 16.3.1.2.2/16-16 |

**Table 16-3. PCI Memory-Mapped Register Map (continued)**

| Offset | Register | Access | Reset | Section/page |
|---|---|---|---|---|
| 0xC08 | Reserved | — | — | |
| 0xC0C | Reserved | — | — | |
| 0xC10 | POWAR0—PCI outbound window 0 (default) attributes register | R/W | 0x8004_401F | 16.3.1.2.4/16-17 |
| 0xC14–0xC1C | Reserved | — | — | |
| 0xC20–0xC3C—Outbound Window 1 | | | | |
| 0xC20 | POTAR1—PCI outbound window 1 translation address register | R/W | 0x0000_0000 | 16.3.1.2.1/16-16 |
| 0xC24 | POTEAR1—PCI outbound window 1 translation extended address register | R/W | 0x0000_0000 | 16.3.1.2.2/16-16 |
| 0xC28 | POWBAR1—PCI outbound window 1 base address register | R/W | 0x0000_0000 | 16.3.1.2.3/16-17 |
| 0xC2C | Reserved | — | — | |
| 0xC30 | POWAR1—PCI outbound window 1 attributes register | R/W | 0x0000_0000 | 16.3.1.2.4/16-17 |
| 0xC34–0xC3C | Reserved | — | — | |
| 0xC40–0xC5C—Outbound Window 2 | | | | |
| 0xC40 | POTAR2—PCI outbound window 2 translation address register | R/W | 0x0000_0000 | 16.3.1.2.1/16-16 |
| 0xC44 | POTEAR2—PCI outbound window 2 translation extended address register | R/W | 0x0000_0000 | 16.3.1.2.2/16-16 |
| 0xC48 | POWBAR2—PCI outbound window 2 base address register | R/W | 0x0000_0000 | 16.3.1.2.3/16-17 |
| 0xC4C | Reserved | — | — | |
| 0xC50 | POWAR2—PCI outbound window 2 attributes register | R/W | 0x0000_0000 | 16.3.1.2.4/16-17 |
| 0xC54–0xC5C | Reserved | — | — | |
| 0xC60–0xC7C—Outbound Window 3 | | | | |
| 0xC60 | POTAR3—PCI outbound window 3 translation address register | R/W | 0x0000_0000 | 16.3.1.2.1/16-16 |
| 0xC64 | POTEAR3—PCI outbound window 3 translation extended address register | R/W | 0x0000_0000 | 16.3.1.2.2/16-16 |
| 0xC68 | POWBAR3—PCI outbound window 3 base address register | R/W | 0x0000_0000 | 16.3.1.2.3/16-17 |
| 0xC6C | Reserved | — | — | |
| 0xC70 | POWAR3—PCI outbound window 3 attributes register | R/W | 0x0000_0000 | 16.3.1.2.4/16-17 |
| 0xC74–0xC7C | Reserved | — | — | |
| 0xC80–0xC9C—Outbound Window 4 | | | | |
| 0xC80 | POTAR4—PCI outbound window 4 translation address register | R/W | 0x0000_0000 | 16.3.1.2.1/16-16 |
| 0xC84 | POTEAR4—PCI outbound window 4 translation extended address register | R/W | 0x0000_0000 | 16.3.1.2.2/16-16 |
| 0xC88 | POWBAR4—PCI outbound window 4 base address register | R/W | 0x0000_0000 | 16.3.1.2.3/16-17 |
| 0xC8C | Reserved | — | — | |
| 0xC90 | POWAR4—PCI outbound window 4 attributes register | R/W | 0x0000_0000 | 16.3.1.2.4/16-17 |

Table 16-3. PCI Memory-Mapped Register Map (continued)

| Offset | Register | Access | Reset | Section/page |
|---|---|---|---|---|
| 0xC94–0xD9C | Reserved | — | — | |
| 0xDA0–0xDBC–Inbound Window 3 | | | | |
| 0xDA0 | PITAR3—PCI inbound window 3 translation address register | R/W | 0x0000_0000 | 16.3.1.3.1/16-20 |
| 0xDA4 | Reserved | — | — | |
| 0xDA8 | PIWBAR3—PCI inbound window 3 base address register | R/W | 0x0000_0000 | 16.3.1.3.2/16-20 |
| 0xDAC | PIWBEAR3—PCI inbound window 3 base extended address register | R/W | 0x0000_0000 | 16.3.1.3.3/16-21 |
| 0xDB0 | PIWAR3—PCI inbound window 3 attributes register | R/W | 0x0000_0000 | 16.3.1.3.4/16-21 |
| 0xDB4–0xDBC | Reserved | — | — | |
| 0xDC0–0xDDC–Inbound Window 2 | | | | |
| 0xDC0 | PITAR2—PCI inbound window 2 translation address register | R/W | 0x0000_0000 | 16.3.1.3.1/16-20 |
| 0xDC4 | Reserved | — | — | |
| 0xDC8 | PIWBAR2—PCI inbound window 2 base address register | R/W | 0x0000_0000 | 16.3.1.3.2/16-20 |
| 0xDCC | PIWBEAR2—PCI inbound window 2 base extended address register | R/W | 0x0000_0000 | 16.3.1.3.3/16-21 |
| 0xDD0 | PIWAR2—PCI inbound window 2 attributes register | R/W | 0x0000_0000 | 16.3.1.3.4/16-21 |
| 0xDD4–0xDDC | Reserved | — | — | |
| 0xDE0–0xDFC–Inbound Window 1 | | | | |
| 0xDE0 | PITAR1—PCI inbound window 1 translation address register | R/W | 0x0000_0000 | 16.3.1.3.1/16-20 |
| 0xDE4 | Reserved | — | — | |
| 0xDE8 | PIWBAR1—PCI inbound window 1 base address register | R/W | 0x0000_0000 | 16.3.1.3.2/16-20 |
| 0xDEC | Reserved | — | — | |
| 0xDF0 | PIWAR1—PCI inbound window 1 attributes register | R/W | 0x0000_0000 | 16.3.1.3.4/16-21 |
| 0xDF4–0xDFC | Reserved | — | — | |
| **PCI Error Management Registers** | | | | |
| 0xE00 | ERR_DR—PCI error detect register | w1c | 0x0000_0000 | 16.3.1.4.1/16-24 |
| 0xE04 | ERR_CAP_DR—PCI error capture disabled register | R/W | 0x0000_0000 | 16.3.1.4.2/16-25 |
| 0xE08 | ERR_EN—PCI error enable register | R/W | 0x0000_0000 | 16.3.1.4.3/16-26 |
| 0xE0C | ERR_ATTRIB—PCI error attributes capture register | R/W | 0x0000_0000 | 16.3.1.4.4/16-27 |
| 0xE10 | ERR_ADDR—PCI error address capture register | R/W | 0x0000_0000 | 16.3.1.4.5/16-28 |
| 0xE14 | ERR_EXT_ADDR—PCI error extended address capture register | R/W | 0x0000_0000 | 16.3.1.4.6/16-28 |
| 0xE18 | ERR_DL—PCI error data low capture register | R/W | 0x0000_0000 | 16.3.1.4.7/16-29 |
| 0xE1C | ERR_DH—PCI error data high capture register | R/W | 0x0000_0000 | 16.3.1.4.8/16-29 |
| 0xE20 | GAS_TIMR—PCI gasket timer register | R/W | 0x0100_3FFF | 16.3.1.4.9/16-29 |

Table 16-3. PCI Memory-Mapped Register Map (continued)

| Offset | Register | Access | Reset | Section/page |
|---|---|---|---|---|
| 0xE28–<br>0xEFC | Reserved | — | — | |
| 0xF00–<br>0xFFC | Reserved for debug | — | — | |

#### 16.3.1.1 PCI Configuration Access Registers

The PCI configuration header, shown in Figure 16-24 and Figure 16-58, is accessed through an indirect method utilizing a pair of 32-bit memory-mapped access registers. For PCI, CFG_ADDR is at offset 0x000 and CFG_DATA is at offset 0x004.

##### 16.3.1.1.1 PCI Configuration Address Register (CFG_ADDR)

The CFG_ADDR register is shown in Figure 16-3.

Offset 0x000 Access: Read/Write

| | 0 | 1 … 7 | 8 … 15 | 16 … 20 | 21 … 23 | 24 … 29 | 30 31 |
|---|---|---|---|---|---|---|---|
| R<br>W | EN | — | Bus Number | Device Number | Function Number | Register Number | — |
| Reset | All zeros | | | | | | |

Figure 16-3. PCI CFG_ADDR Register

Table 16-4 describes the bit settings for the CFG_ADDR register.

Table 16-4. PCI CFG_ADDR Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0 | Enable | Allow a PCI configuration access when PCI CFG_DATA is accessed |
| 1–7 | — | Reserved |
| 8–15 | Bus Number | PCI bus number to access |
| 16–20 | Device Number | Device number to access on specified bus |
| 21–23 | Function Number | Function to access within specified device |
| 24–29 | Register Number | 32-bit register to access within specified device |
| 30–31 | — | Reserved, hardwired to logic 00 |

Bus number 0xb00 and device number 0b0_0000 are used to configure the internal PCI configuration header of the PCI controller itself.

See Section 16.4.2.11.2, “Host Accessing the PCI Configuration Space,” and Section 16.4.2.11.3, “Agent Accessing the PCI Configuration Space,” for usage of PCI CFG_ADDR.

#### 16.3.1.1.2 PCI Configuration Data Register (CFG_DATA)

The CFG_DATA register is shown in Figure 16-3.

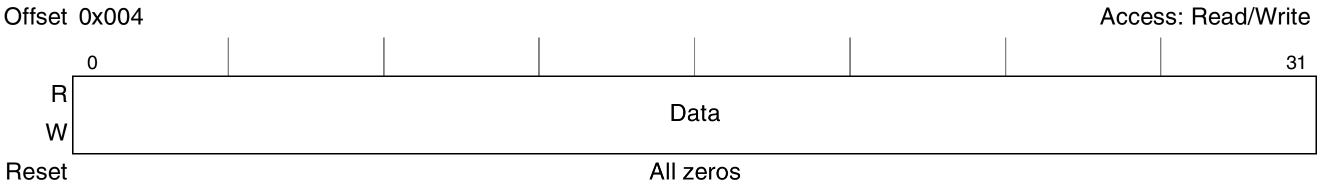


**Figure 16-4. PCI CFG_DATA Register**

Table 16-5 describes the bit settings for the CFG_DATA register

**Table 16-5. PCI CFG_DATA Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–31 | Data | A read or write to this register starts a PCI configuration cycle if the PCI CFG_ADDR enable bit is set. If the enable bit is not set, a PCI I/O transaction is generated. |

The CFG_DATA register is a 4-byte window into the little-endian PCI configuration header data structure; therefore, byte addressing within the CFG_DATA register uses little-endian convention. Note that CFG_DATA may contain 1, 2, 3, or 4 bytes depending on the size of the register being accessed.

See Section 16.4.2.11.2, “Host Accessing the PCI Configuration Space,” and Section 16.4.2.11.3, “Agent Accessing the PCI Configuration Space,” for usage of CFG_DATA.

#### 16.3.1.1.3 PCI Interrupt Acknowledge Register (INT_ACK)

An external PCI interrupt acknowledge transaction is generated by reading the INT_ACK register. For PCI, INT_ACK is at offset 0x008. INT_ACK is shown in Figure 16-5.

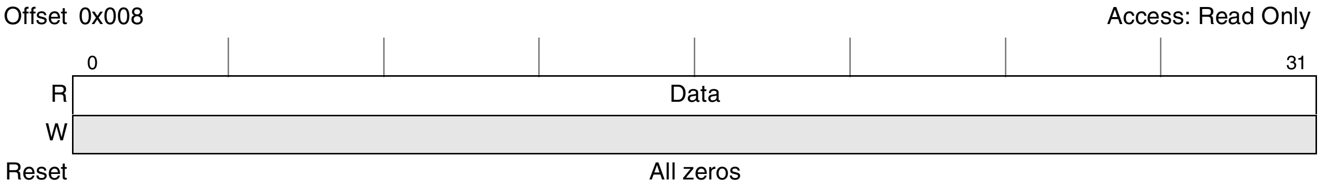


**Figure 16-5. PCI INT_ACK Register**

Table 16-6 describes the bit settings for the INT_ACK register.

**Table 16-6. PCI INT_ACK Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–31 | Data | A read to this register generates a PCI interrupt acknowledge cycle. |

### 16.3.1.2 PCI ATMU Outbound Registers

The outbound address translation and mapping unit controls the mapping of transactions from the internal platform address space to the external PCI address space. The outbound ATMU consists of four translation windows plus a default translation for transactions that do not hit in one of the four windows.

Each window contains a base address that points to the beginning of the window in the local address map, a translation address that specifies the high-order bits of the transaction in the external PCI address space, and a set of attributes including window size and external transaction type.

Each window must be aligned based on the granularity specified by the window size. If two outbound ATMU windows overlap in the local address space, the mapping of the lower numbered window has precedence over the higher numbered window.

Window 0 is the default window and is the only window enabled upon reset. The default outbound register set is used when a transaction misses in all of the other outbound windows.

#### 16.3.1.2.1 PCI Outbound Translation Address Registers (POTAR*n*)

The PCI outbound translation address registers (POTAR*n*) select the starting addresses in the PCI address space for hits in the PCI outbound windows. The translated address is created by concatenating the transaction offset to this translation address. The format of the POTAR*n* is shown in Figure 16-6.

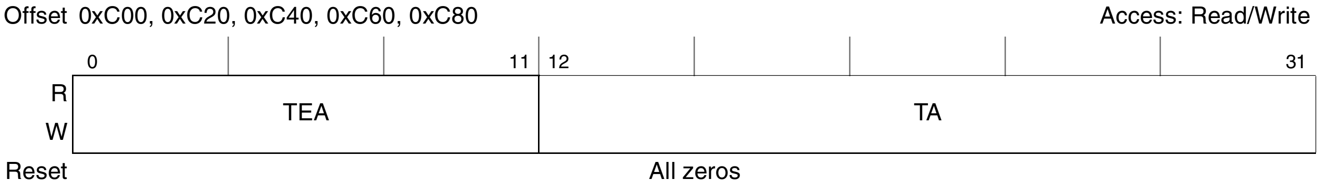


**Figure 16-6. PCI Outbound Translation Address Registers (POTAR*n*)**

Table 16-7 describes the fields of the POTAR*n* registers.

**Table 16-7. POTAR*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–11 | TEA | Translation extended address. Represents bits [43:32] of a 64-bit PCI address (bit 0 is lsb). |
| 12–31 | TA | Translation address. Represents bits [31:12] of the PCI address. The specified address must be aligned to the window size, as defined by POWAR*n*[OWS]. |

#### 16.3.1.2.2 PCI Outbound Translation Extended Address Registers (POTEAR*n*)

The PCI outbound translation extended address registers (POTEAR*n*) contain the most significant bits of a 64-bit translation address. The format of POTEAR*n* is shown in Figure 16-7.

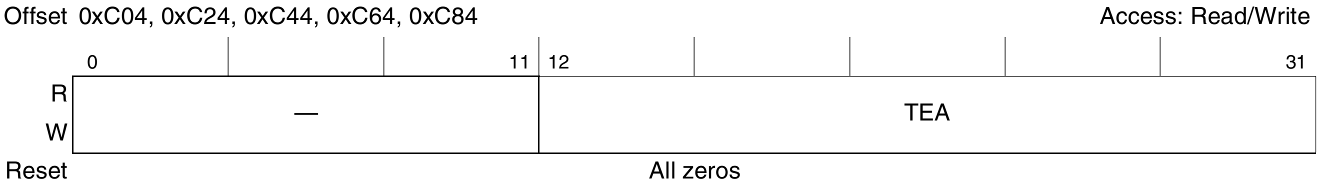


**Figure 16-7. PCI Outbound Translation Extended Address Registers (POTEAR*n*)**

Table 16-8 describes the fields of the POTEAR*n*.

Table 16-8. POTEAR*n* Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–11 | — | Reserved |
| 12–31 | TEA | Translation extended address. Comprise bits [63:44] of the translation address. |

### 16.3.1.2.3 PCI Outbound Window Base Address Registers (POWBAR*n*)

The PCI outbound window base address registers (POWBAR*n*) point to the beginning of each translation window in the local 32-bit address space. Addresses for outbound transactions are compared to the appropriate bits in these registers, according to the sizes of the windows. If a transaction does not fall within one of these windows, the default translation and mapping is used. The default window is always enabled and used when the other windows miss.

Note that POWBAR0 (for outbound ATMU window 0) is not used, because window 0 is the default window used when no other windows match. POWBAR0 may be read from and written to, but the value is ignored.

The format of the POWBAR*n* is shown in Figure 16-8.

Offset 0xC28, 0xC48, 0xC68, 0xC88 Access: Read/Write

| | 0 – 11 | 12 – 31 |
|---|---|---|
| R<br>W | WBEA | WBA |
| Reset | All zeros | |

Figure 16-8. PCI Outbound Window Base Address Registers (POWBAR*n*)

Table 16-9 describes the field of the POWBAR*n*.

Table 16-9. POWBAR*n* Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–11 | WBEA | Window base extended address. Bits 0–7 are reserved; bits 8–11 correspond to bits [0:3] of the (internal platform) base address.<br>0x000 – 0x00F are valid.<br>0x010 and greater are reserved. |
| 12–31 | WBA | Window base address. Source address which is the starting point for the outbound window.<br>The specified address must be aligned to the window size, as defined by POWAR*n*[OWS]. Corresponds to bits [4-35] of the (internal platform) base address. |

### 16.3.1.2.4 PCI Outbound Window Attributes Registers (POWAR*n*)

The PCI outbound window attributes registers (POWAR*n*) define the window sizes to translate and other attributes for the translations. The minimum window size is 4 Kbytes. The maximum window size is 16 Gbytes.

The default window attribute register, POWAR0, is shown in Figure 16-9. Note that the fields for all of the POWAR*n* registers are the same, only the reset values are different.

Offset 0xC10 Access: Read/Write

| | 0 | 1–11 | 12–15 | 16–19 | 20–25 | 26–31 |
|---|---|---|---|---|---|---|
| R<br>W | EN | — | RTT | WTT | — | OWS |
| Reset | 1 | 0 0 0 0 0 0 0 0 0 0 0 | 0 1 0 0 | 0 1 0 0 | 0 0 0 0 0 0 | 0 1 1 1 1 1[1] |

**Figure 16-9. PCI Outbound Window 0 (Default) Attributes Register (POWAR0)**

[1] The default window is enabled, configured for memory read and memory write, and set to an OWS size of 4 Gbytes.

POWAR1–POWAR4 are shown in Figure 16-10.

Offset 0xC30, 0xC50, 0xC70, 0xC90 Access: Read/Write

| | 0 | 1–11 | 12–15 | 16–19 | 20–25 | 26–31 |
|---|---|---|---|---|---|---|
| R<br>W | EN | — | RTT | WTT | — | OWS |
| Reset | All zeros | | | | | |

**Figure 16-10. PCI Outbound Window 1–4 Attributes Registers (POWAR1–POWAR4)**

Table 16-10 describes the fields for the POWAR*n* registers.

**Table 16-10. POWAR*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | EN | Enable. Enables this address translation |
| 1–11 | — | Reserved |
| 12–15 | RTT | Read transaction type to run on PCI<br>0000 Reserved<br>...<br>0011 Reserved<br>0100 Memory Read<br>0101 Reserved<br>...<br>0111 Reserved<br>1000 I/O Read<br>1001 Reserved<br>...<br>1111 Reserved |
| 16–19 | WTT | Write transaction type to run on PCI<br>0000 Reserved<br>...<br>0011 Reserved<br>0100 Memory Write<br>0101 Reserved<br>...<br>0111 Reserved<br>1000 I/O Write<br>1001 Reserved<br>...<br>1111 Reserved |

**Table 16-10. POWAR*n* Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 20–25 | — | Reserved |
| 26–31 | OWS | Outbound window size. Outbound translation window size N which is the encoded 2^(N+1) bytes window size. The smallest window size is 4 Kbytes.<br>000000Reserved<br>...<br>0010114-Kbyte window size<br>0011008-Kbyte window size<br>...<br>0111114-Gbyte window size<br>1000008-Gbyte window size<br>10000116-Gbyte window size<br>100010Reserved<br>...<br>111111Reserved<br>The default POWAR register (0xC10) has an OWS value of 011111. |

### 16.3.1.3 PCI ATMU Inbound Registers

The inbound address translation and mapping unit controls the mapping of transactions from the external PCI address space to the internal platform address space. The inbound ATMU is comprised of four windows—a configuration window and three general translation windows. The configuration window has higher priority than all other inbound ATMU windows and takes precedence over them if there is an overlap.

Each window contains the following:

- A base address, which points to the beginning of the window in the external PCI address map. The base address of each window is also accessible by PCI configuration transactions as base address registers within the PCI configuration header, as shown in Figure 16-26. The registers may be read or updated equivalently through the ATMU memory map or through PCI configuration transactions to the PCI configuration header.
- A translation address, which specifies the upper order bits of the transaction in the local address space.
- A set of attributes including window size and internal transaction attributes.

Each window's base address and translation address must be aligned to the size of the window. If two general inbound ATMU windows overlap in the external PCI address space, the mappings of the lower numbered window are applied; PCSRBAR takes priority over any overlapping inbound ATMU window. In addition, if inbound ATMU windows are overlapped, the ATMU windows must not map to the same address with different sets of attributes (other than window size).

Note that PCSRBAR in the PCI configuration header acts as a fourth inbound window that translates a 1-Mbyte region of PCI space to the local configuration space pointed to by CCSRBAR. PCSRBAR can be accessed by PCI configuration cycles or by accessing the PCI configuration header through the PCI CFG_ADDR and PCI CFG_DATA registers. See Section 16.3.1.1.1, "PCI Configuration Address Register (CFG_ADDR)," Section 16.3.1.1.2, "PCI Configuration Data Register (CFG_DATA)," and

Section 16.3.2.11, “PCI Base Address Registers.” All accesses to PCSRBAR have an automatic internal byte lane redirection from the little-endian PCI bus to the big-endian CCSRBAR configuration space.

#### 16.3.1.3.1 PCI Inbound Translation Address Registers (PITAR*n*)

The PCI inbound translation address registers (PITAR*n*) points to the beginning of the local address space for the inbound window. The translated address is created by concatenating the transaction offset to this translation address. The format of the PITAR*n* is shown in Figure 16-11.

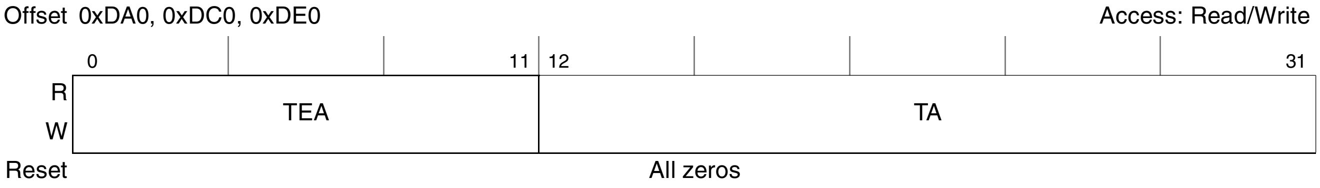


**Figure 16-11. PCI Inbound Translation Address Registers (PITAR*n*)**

Table 16-11 describes the fields of the PITAR*n* registers

**Table 16-11. PITAR*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–11 | TEA | Translation extended address. Bits 0–7 are reserved; bits 8–11 correspond to bits [0:3] of the local translation address.<br>0x000 – 0x00F are valid.<br>0x010 and greater are reserved. |
| 12–31 | TA | Translation address. Indicates the starting point of the inbound translated address. The specified address must be aligned to the window size, as defined by PIWAR*n*[IWS]. TA corresponds to bits [4:23] of the 36-bit local translation address. |

#### 16.3.1.3.2 PCI Inbound Window Base Address Registers (PIWBAR*n*)

The PCI inbound window base address registers (PIWBAR*n*) select the PCI base address for the windows that are translated to the internal platform address space. Addresses for inbound transactions are compared to these windows. If a PCI transaction does not fall within one of these spaces, then the PCI interface does not assert DEVSEL. The PIWBAR*n* is shown in Figure 16-12.

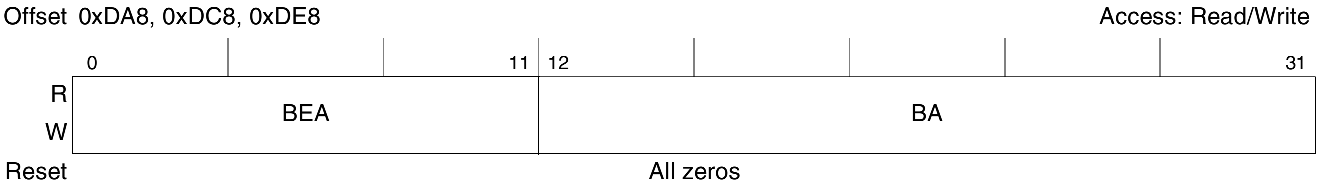


**Figure 16-12. PCI Inbound Window Base Address Registers**

Table 16-12 describes the fields of the PIWBAR*n* registers.

Table 16-12. PIWBAR Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–11 | BEA | Base extended address. Corresponds to bits 43–32 of a 64-bit PCI base address. |
| 12–31 | BA | Base address. Corresponds to bits 31–12 of a PCI base address. The specified address must be aligned to the window size, as defined by PIWAR*n*[IWS]. |

### 16.3.1.3.3 PCI Inbound Window Base Extended Address Registers (PIWBEAR*n*)

The PCI inbound window base extended address registers (PIWBEAR*n*) contain the most-significant bits of a 64-bit base address. Note that inbound window 1 supports only a 32-bit base address and does not define an inbound window base extended address register. The PIWBEAR*n* are shown in Figure 16-13.

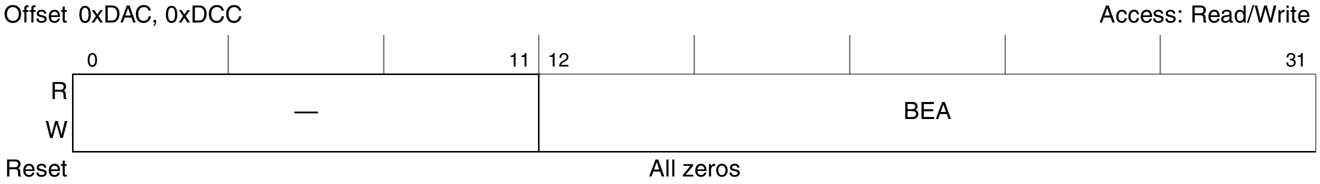


Figure 16-13. PCI Inbound Window Base Extended Address Registers (PIWBEAR*n*)

Table 16-13 describes the fields of the PIWBEAR*n* registers.

Table 16-13. PIWBEAR Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–11 | — | Reserved |
| 12–31 | BEA | Base extended address. Corresponds to bits 63–44 of a 64-bit PCI base address. |

### 16.3.1.3.4 PCI Inbound Window Attributes Registers (PIWAR*n*)

The PCI inbound window attributes registers (PIWAR*n*) define the window sizes to translate and other attributes for the translations. 16 Gbytes is the largest window size allowed. The format of the PIWAR*n* is shown in Figure 16-14.

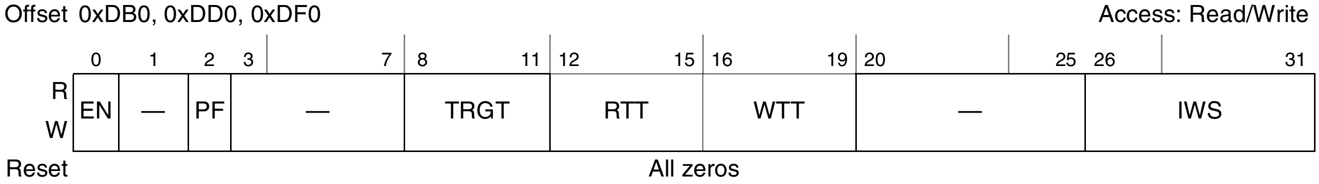


Figure 16-14. PCI Inbound Window Attributes Registers

Table 16-14 describes the fields of the PIWAR*n* registers.

**Table 16-14. PIWAR*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | EN | Enable. Enables this address translation |
| 1 | — | Reserved |
| 2 | PF | Prefetchable. Indicates that the address space is prefetchable so that prefetching and streaming are attempted.<br>0 Not prefetchable<br>1 Prefetchable |
| 3–7 | — | Reserved |
| 8–11 | TRGT | 0000 Reserved<br>0001 PCI Express 2<br>0010 PCI Express 1<br>0011 PCI Express 3<br>0100–1110 Reserved<br>1111 Local memory space |
| 12–15 | RTT | Read transaction type. Transaction type to run if access is a read. The field description differs subject to the transaction being targeted to I/O interface or to local memory.<br>Following are the transaction type settings for reads to an I/O interface:<br>0000–0011 Reserved<br>0100 Read<br>0101–1111 Reserved<br>Following are the transaction type settings for reads to local memory:<br>0000–0011 Reserved<br>0100 Read, don’t snoop local processor<br>0101 Read, snoop local processor<br>0110 Reserved<br>0111 Read, unlock L2 cache line<br>1000–1111 Reserved |
| 16–19 | WTT | Write transaction type. Transaction type to run if access is a write. The field description differs subject to the transaction being targeted to an I/O interface or to local memory.<br>Following are the transaction type settings for writes to an I/O interface:<br>0000–0011 Reserved<br>0100 Write<br>0101–1111 Reserved<br>Following are the transaction type settings for writes to local memory:<br>0000–Reserved<br>0100 Write, don’t snoop local processor<br>0101 Write, snoop local processor<br>0110 Write, allocate L2 cache line<br>0111 Write, allocate and lock L2 cache line<br>1000–1111 Reserved |

Table 16-14. PIWAR*n* Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 20–25 | — | Reserved |
| 26–31 | IWS | Inbound window size. Inbound translation window size N which is the encoded 2^(N+1) bytes window size. The smallest window is 4 Kbytes.<br>000000 Reserved<br>...<br>001011 4-Kbyte window size<br>001100 8-Kbyte window size<br>...<br>011111 4-Gbyte window size<br>100000 8-Gbyte window size<br>100001 16-Gbyte window size<br>100010 Reserved<br>...<br>111111 Reserved<br>For configuration and run-time registers, the window size is fixed at<br>010011 1-Mbyte window size<br>For register set 0, the window size is limited to 4 Gbytes or smaller. |

#### 16.3.1.4 PCI Error Management Registers

When a PCI error is detected, the appropriate error bit is set in the PCI error detect register. Subsequent errors set the appropriate error bits in the error detection registers, but relevant information (attributes, address, and data) is captured only for the first error. The PCI error detect register is a write-1-to-clear type register. That is, reading from this register occurs normally; however, write operations are different in that the bits can be cleared but not set. A bit is cleared whenever the register is written, and the data in the corresponding bit location is a 1. For example, to clear bit 25 and not affect any other bits in the register, the value 0x0000_0040 is written to the register.

The error bit is set regardless of the state of the corresponding error enable bit in the PCI error enable register. The error enable bits are used to send or block the error reporting to the interrupt mechanism. The interrupt can be cleared by writing 0xFFFF_FFFF to the PCI error detect register.

A master-abort condition during a configuration cycle is not necessarily an error. In this case, if relevant, the master abort error enable can be disabled to prevent the reporting of master-aborts during outbound configuration cycles. Master-aborts during configuration reads return 0xFFFF_FFFF.

For an inbound configuration write transaction with a parity error, the device always updates the register access and generates the error interrupt if the interrupt enabled bit is set.

See Section 16.4.2.13, “PCI Error Functions,” for more detail on error handling.

#### 16.3.1.4.1 PCI Error Detect Register (ERR_DR)

Offset 0xE00 Access: w1c

| | 0 | 1–7 |
|---|---|---|
| R | Multiple PCI Errors | — |
| W | w1c | |
| Reset | All zeros | |

| | 8–15 |
|---|---|
| R | — |
| W | |
| Reset | All zeros |

| | 16–20 | 21 | 22 | 23 |
|---|---|---|---|---|
| R | — | Addr Parity error | Rcvd $\overline{\text{SERR}}$ error | Mstr $\overline{\text{PERR}}$ error |
| W | | w1c | w1c | w1c |
| Reset | All zeros | | | |

| | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|
| R | Trgt $\overline{\text{PERR}}$ error | Mstr abort error | Trgt abort error | OWMSV error | ORMSV error | IRMSV error | SCM error | TOE error |
| W | w1c | w1c | w1c | w1c | w1c | w1c | w1c | w1c |
| Reset | All zeros | | | | | | | |

**Figure 16-15. PCI Error Detect Register (ERR_DR)**

Table 16-15 describes ERR_DR fields. Note that uncorrectable read errors may cause the assertion of *core_fault_in*, which causes the core to generate a machine check interrupt, unless it is disabled (by clearing HID1[RFXE]). If RFXE is zero and an error occurs, the appropriate parity detect and master-abort bits in ERR_DR must be cleared and the appropriate enable bits in ERR_EN must be set to ensure that an interrupt is generated. See Section 6.10.2, “Hardware Implementation-Dependent Register 1 (HID1).”

**Table 16-15. ERR_DR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | Multiple PCI errors | 0 Multiple PCI errors of the same type were not detected (write-1-to-clear)<br>1 Multiple PCI errors of the same type were detected |
| 1–20 | — | Reserved |
| 21 | Addr Parity error | Address parity error (write-1-to-clear) |
| 22 | Rcvd $\overline{\text{SERR}}$ error | Received $\overline{\text{SERR}}$ error (write-1-to-clear) |
| 23 | Mstr $\overline{\text{PERR}}$ error | Master $\overline{\text{PERR}}$ error (write-1-to-clear) |
| 24 | Trgt $\overline{\text{PERR}}$ error | Target $\overline{\text{PERR}}$ error (write-1-to-clear) |
| 25 | Mstr abort error | Master abort error (write-1-to-clear) |
| 26 | Trgt abort error | Target abort error (write-1-to-clear) |

Table 16-15. ERR_DR Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 27 | OWMSV error | Outbound write memory space violation error (write-1-to-clear) |
| 28 | ORMSV error | Outbound read memory space violation error (write-1-to-clear) |
| 29 | IRMSV error | Inbound read memory space violation error (write-1-to-clear) |
| 30 | SCM error | Split completion message error (write-1-to-clear) |
| 31 | TOE error | Time-out error (write-1-to-clear) |

#### 16.3.1.4.2 PCI Error Capture Disable Register (ERR_CAP_DR)

Offset 0xE04 Access: Read/Write

| | 0 … 7 |
|---|---|
| R<br>W | — |
| Reset | All zeros |

| | 8 … 15 |
|---|---|
| R<br>W | — |
| Reset | All zeros |

| | 16 … 20 | 21 | 22 | 23 |
|---|---|---|---|---|
| R<br>W | — | Addr parity error capture disable | Rcvd $\overline{\text{SERR}}$ error capture disable | Mstr $\overline{\text{PERR}}$ error capture disable |
| Reset | All zeros | | | |

| | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|
| R<br>W | Trgt $\overline{\text{PERR}}$ error capture disable | Mstr abort error capture disable | Trgt abort error capture disable | OWMSV error capture disable | ORMSV error capture disable | IRMSV error capture disable | SCM error capture disable | TOE error capture disable |
| Reset | All zeros | | | | | | | |

Figure 16-16. PCI Error Capture Disable Register (ERR_CAP_DR)

Table 16-16. ERR_CAP_DR Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–20 | — | Reserved |
| 21 | Addr parity error capture disable | Disable capture for address parity errors |
| 22 | Rcvd $\overline{\text{SERR}}$ error capture disable | Disable capture for received $\overline{\text{SERR}}$ errors |
| 23 | Mstr $\overline{\text{PERR}}$ error capture disable | Disable capture for master $\overline{\text{PERR}}$ errors |
| 24 | Trgt $\overline{\text{PERR}}$ error capture disable | Disable capture for target $\overline{\text{PERR}}$ errors |
| 25 | Mstr abort error capture disable | Disable capture for master abort errors |

Table 16-16. ERR_CAP_DR Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 26 | Trgt abort error capture disable | Disable capture for target abort errors |
| 27 | OWMSV error capture disable | Disable capture for outbound write memory space violation errors |
| 28 | ORMSV error capture disable | Disable capture for outbound read memory space violation errors |
| 29 | IRMSV error capture disable | Disable capture for inbound read memory space violation errors |
| 30 | SCM error capture disable | Disable capture for split completion message errors |
| 31 | TOE error capture disable | Disable capture for time-out errors |

### 16.3.1.4.3 PCI Error Enable Register (ERR_EN)

Offset 0xE08 Access: Read/Write

| | 0–7 |
|---|---|
| R<br>W | — |
| Reset | All zeros |

| | 8–15 |
|---|---|
| R<br>W | — |
| Reset | All zeros |

| | 16–20 | 21 | 22 | 23 |
|---|---|---|---|---|
| R<br>W | — | Addr parity error enable | Rcvd $\overline{\text{SERR}}$ error enable | Mstr $\overline{\text{PERR}}$ error enable |
| Reset | All zeros | | | |

| | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|
| R<br>W | Trgt $\overline{\text{PERR}}$ error enable | Mstr abort error enable | Trgt abort error enable | OWMSV error enable | ORMSV error enable | IRMSV error enable | SCM error enable | TOE error enable |
| Reset | All zeros | | | | | | | |

Figure 16-17. PCI Error Enable Register (ERR_EN)

Table 16-17 describes ERR_EN fields. Note that uncorrectable read errors may cause the assertion of *core_fault_in*, which causes the core to generate a machine check interrupt, unless it is disabled (by clearing HID1[RFXE]). If RFXE is zero and this error occurs, the appropriate parity detect and master-abort bits in ERR_DR must be cleared and the appropriate enable bits in ERR_EN must be set to ensure that an interrupt is generated. For more information, see Section 6.10.2, “Hardware Implementation-Dependent Register 1 (HID1).”

Table 16-17. ERR_EN Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–20 | — | Reserved |
| 21 | Addr parity error enable | Enable reporting address parity errors |
| 22 | Rcvd $\overline{\text{SERR}}$ error enable | Enable reporting received $\overline{\text{SERR}}$ errors |
| 23 | Mstr $\overline{\text{PERR}}$ error enable | Enable reporting master $\overline{\text{PERR}}$ errors |
| 24 | Trgt $\overline{\text{PERR}}$ error enable | Enable reporting target $\overline{\text{PERR}}$ errors |
| 25 | Mstr abort error enable | Enable reporting master abort errors |
| 26 | Trgt abort error enable | Enable reporting target abort errors |
| 27 | OWMSV error enable | Enable reporting outbound write memory space violation errors |
| 28 | ORMSV error enable | Enable reporting outbound read memory space violation errors |
| 29 | IRMSV error enable | Enable reporting inbound read memory space violation errors |
| 30 | SCM error enable | Enable reporting split completion message errors |
| 31 | TOE error enable | Enable reporting time-out errors |

### 16.3.1.4.4 PCI Error Attributes Capture Register (ERR_ATTRIB)

Offset 0xE0C Access: Read/Write

| | 0–3 | 4–7 | 8 | 9 | 10 | 11–15 | 16–19 | 20–30 | 31 |
|---|---|---|---|---|---|---|---|---|---|
| R<br>W | High word byte enables | Low word byte enables | High parity bit | Low parity bit | — | Error source | Command | — | Valid |
| Reset | All zeros | | | | | | | | |

Figure 16-18. PCI Error Attributes Capture Register (ERR_ATTRIB)

Table 16-18. ERR_ATTRIB Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–3 | High word byte enables | PCI byte enables for most significant word of the double word |
| 4–7 | Low word byte enables | PCI byte enables for least significant word of the double word |
| 8 | High parity bit | Parity bit for most significant PCI bus data word (only valid for 64-bit PCI bus) |
| 9 | Low parity bit | Parity bit for least significant PCI bus data word |
| 10 | — | Reserved |

Table 16-18. ERR_ATTRIB Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 11–15 | Error source | The source of the PCI transaction<br>00000 PCI<br>00001 PCI Express 2<br>00010 PCI Express 1<br>00011 PCI Express 3<br>00101 USB 1, USB2, or USB3<br>01101 SATA 1 or SATA 2<br>01010 Boot sequencer<br>01011 eSDHC<br>10000 Processor instruction<br>10001 Processor data<br>10101 DMA<br>11000 eTSEC1 or Security<br>11010 eTSEC3<br>All other settings reserved |
| 16–19 | Command | PCI command |
| 20–30 | — | Reserved |
| 31 | Valid info | The PCI bus capture registers contain valid information |

### 16.3.1.4.5 PCI Error Address Capture Register (ERR_ADDR)

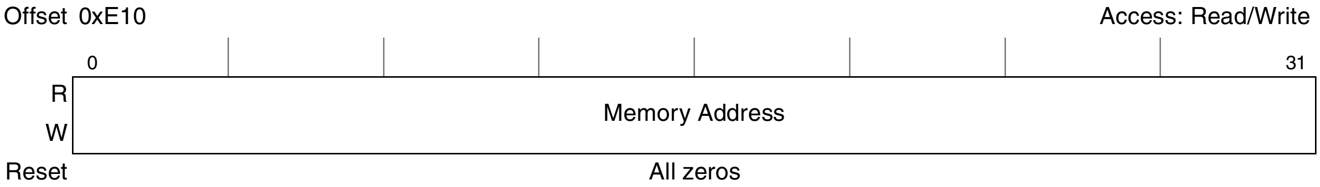


Figure 16-19. PCI Error Address Capture Register (ERR_ADDR)

Table 16-19. ERR_ADDR Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–31 | Memory address | Memory transaction address |

### 16.3.1.4.6 PCI Error Extended Address Capture Register (ERR_EXT_ADDR)

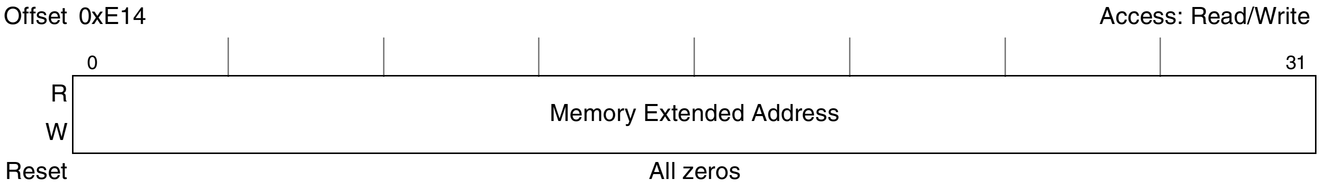


Figure 16-20. PCI Error Extended Address Capture Register (ERR_EXT_ADDR)

Table 16-20. ERR_EXT_ADDR Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–31 | Memory extended address | Memory transaction extended address |

#### 16.3.1.4.7 PCI Error Data Low Capture Register (ERR_DL)

Offset 0xE18 Access: Read/Write

| | 0–31 |
|---|---|
| R<br>W | Data low |
| Reset | All zeros |

**Figure 16-21. PCI Error Data Low Capture Register (ERR_DL)**

**Table 16-21. ERR_DL Field Description**

| Bits | Name | Description |
|---|---|---|
| 0–31 | Data low | Least significant PCI bus data word |

#### 16.3.1.4.8 PCI Error Data High Capture Register (ERR_DH)

Offset 0xE1C Access: Read/Write

| | 0–31 |
|---|---|
| R<br>W | Data high |
| Reset | All zeros |

**Figure 16-22. PCI Error Data High Capture Register (ERR_DH)**

**Table 16-22. ERR_DH Field Description**

| Bits | Name | Description |
|---|---|---|
| 0–31 | Data high | Most significant PCI bus data word (only valid with 64-bit PCI bus) |

#### 16.3.1.4.9 PCI Gasket Timer Register (GAS_TIMR)

Offset 0xE20 Access: Read/Write

| | 0–6 | 7 | 8–31 |
|---|---|---|---|
| R<br>W | — | EN | TCNT |
| Reset | 0 0 0 0 0 0 0 | 1 | 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 |

**Figure 16-23. PCI Gasket Timer Register (GAS_TIMR)**

**Table 16-23. GAS_TIMR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–6 | — | Reserved |
| 7 | EN | Gasket timer enable.<br>0 gasket timer is disabled.<br>1 gasket timer is enabled. |
| 8–31 | TCNT | Number of system clocks to purge a non-prefetchable inbound read buffer |

## 16.3.2 PCI Configuration Header

The *PCI Local Bus Specification* defines the configuration registers contained within the PCI configuration header from 0x00 through 0x3F. Figure 16-24 lists the common PCI configuration header as implemented by the device.

▨ Reserved

| | | | | Address Offset (Hex) |
|---|---|---|---|---|
| Device ID | | Vendor ID | | 00 |
| PCI Bus Status | | PCI Bus Command | | 04 |
| Bus Base Class Code | Subclass Code | Bus Programming Interface | Revision ID | 08 |
| BIST Control | Header Type | Bus Latency Timer | Bus Cache Line Size | 0C |
| PCI Configuration and Status Register Base Address Register (PCSRBAR) | | | | 10 |
| 32-Bit Memory Base Address Register | | | | 14 |
| 64-Bit Low Memory Base Address Register | | | | 18 |
| 64-Bit High Memory Base Address Register | | | | 1C |
| 64-Bit Low Memory Base Address Register | | | | 20 |
| 64-Bit High Memory Base Address Register | | | | 24 |
| ▨ | | | | 28 |
| Subsystem ID | | Subsystem Vendor ID | | 2C |
| ▨ | | | | 30 |
| ▨ | | | PCI Bus Capability Pointer | 34 |
| ▨ | | | | 38 |
| PCI Bus MAX_LAT | PCI Bus MIN_GNT | PCI Bus Interrupt Pin | PCI Bus Interrupt Line | 3C |
| ▨ | | | | 40 |
| PCI Bus Arbiter Configuration | | PCI Bus Function | | 44 |

**Figure 16-24. Common PCI Configuration Header**

Table 16-49 in Section 16.4.2.11.1, “PCI Configuration Space Header,” provides a summary of the PCI configuration header registers. Detailed descriptions of these registers are provided in the *PCI Local Bus Specification*.

### 16.3.2.1 PCI Vendor ID Register—Offset 0x00

The PCI vendor ID register, shown in Figure 16-25, is used to identify the manufacturer of the part.

Offset 0x00

Access: Read only

| | 15 | | | | | | | | | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Vendor ID | | | | | | | | | | | | | | | |
| W | ▨ | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 |

**Figure 16-25. PCI Vendor ID Register**

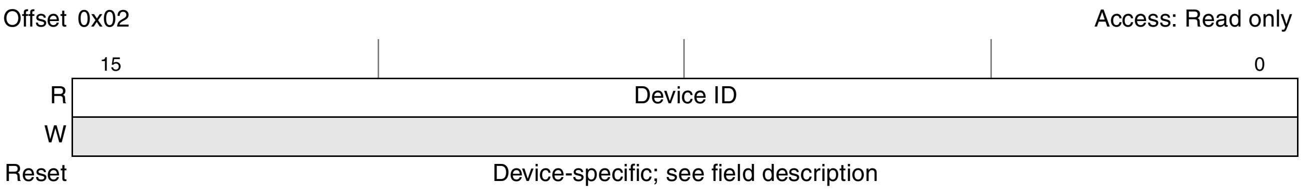
Table 16-24 describes PCI vendor ID register fields.

Table 16-24. PCI Vendor ID Register Field Description

| Bits | Name | Description |
|---|---|---|
| 15–0 | Vendor ID | 0x1957 (Freescale) |

### 16.3.2.2 PCI Device ID Register—Offset 0x02

The PCI device ID register, shown in Figure 16-26, is used to identify the device.



Offset 0x02 Access: Read only

| | 15 … 0 |
|---|---|
| R | Device ID |
| W | |
| Reset | Device-specific; see field description |

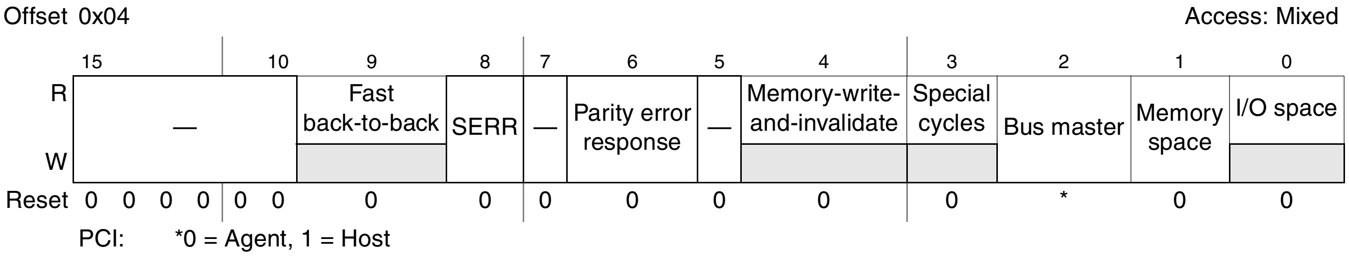
Figure 16-26. PCI Device ID Register

Table 16-25. PCI Device ID Register Field Description

| Bits | Name | Description |
|---|---|---|
| 15–0 | Device ID | 0x0050 MPC8536E<br>**Note:** 0x0051MPC8536 |

### 16.3.2.3 PCI Bus Command Register—Offset 0x04

The 2-byte PCI bus command register provides control over the ability to generate and respond to PCI cycles. Table 16-26 describes the bits of the PCI bus command register.



Offset 0x04 Access: Mixed

| | 15 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | Fast back-to-back | SERR | — | Parity error response | — | Memory-write-and-invalidate | Special cycles | Bus master | Memory space | I/O space |
| Reset | 0 0 0 0 0 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | * | 0 | 0 |

PCI: *0 = Agent, 1 = Host

Figure 16-27. PCI Bus Command Register

Table 16-26. PCI Bus Command Register Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 15–10 | — | Reserved |
| 9 | Fast back-to-back | Hard-wired to 0, indicating that this PCI controller (as a master) does not run fast back-to-back transactions. |

**Table 16-26. PCI Bus Command Register Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 8 | SERR | Controls the $\overline{\text{PCI\_SERR}}$ driver of this PCI controller. This bit (and bit 6) must be set to report address parity errors.<br>0 Disables the $\overline{\text{PCI\_SERR}}$ driver<br>1 Enables the $\overline{\text{PCI\_SERR}}$ driver |
| 7 | — | Reserved |
| 6 | Parity error response | Controls whether this PCI controller responds to parity errors.<br>0 Parity errors are ignored and normal operation continues.<br>1 Parity errors cause the appropriate bit in the PCI status register to be set. However, note that errors are reported based on the values set in the PCI error enable and detection registers. |
| 5 | — | Reserved |
| 4 | Memory-write-and-invalidate | Hard-wired to 0, indicating that this PCI controller, acting as a master, cannot generate the memory-write-and-invalidate command. |
| 3 | Special-cycles | Hard-wired to 0, indicating that this PCI controller (as a target) ignores all special-cycle commands. |
| 2 | Bus master | Indicates whether this PCI controller is configured as a master. This indicates the setting of the host/agent configuration input signal at power-on reset.<br>0 Disables the ability to generate PCI accesses<br>1 Enables this PCI controller to behave as a PCI bus master (Host) |
| 1 | Memory space | Controls whether this PCI controller (as a target) responds to memory accesses.<br>0 This PCI controller does not respond to PCI memory space accesses.<br>1 This PCI controller (as a target) responds to PCI memory space accesses. |
| 0 | I/O space | Hard-wired to 0, indicating that this PCI controller (as a target) does not respond to PCI I/O space accesses. |

### 16.3.2.4 PCI Bus Status Register—Offset 0x06

The 2-byte PCI bus status register is used to record status information for PCI bus bus-related events. The definition of each bit is given in Table 16-27. Only 2-byte accesses to address offset 0x06 are allowed.

Reads to this register behave normally. Writes are slightly different in that bits can be cleared, but not set. A bit is cleared whenever the register is written, and the data in the corresponding bit location is a 1. For example, to clear bit 14 without affecting any other bits in the register, write the value 0b0100_0000_0000_0000 to the register.

Offset 0x06 Access: Mixed

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| R | Detected parity error | Signaled system error | Received master-abort | Received target-abort | Signaled target-abort | DEVSEL timing | | Master data parity error detected |
| W | w1c | w1c | w1c | w1c | w1c | | | w1c |
| Reset | All zeros | | | | | | | |

| | 7 | 6 | 5 | 4 | 3 | | | 0 |
|---|---|---|---|---|---|---|---|---|
| R | Fast back-to-back capable | — | 66-MHz capable | Capabilities list | — | | | |
| W | | | | | | | | |
| Reset | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

**Figure 16-28. PCI Bus Status Register**

**Table 16-27. PCI Bus Status Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 15 | Detected parity error | Set whenever this PCI controller detects a PCI parity error, even if parity error handling is disabled (as controlled by bit 6 in the PCI bus command register). |
| 14 | Signaled system error | Set whenever this PCI controller asserts $\overline{\text{PCI\_SERR}}$. |
| 13 | Received master-abort | Set whenever this PCI controller, acting as the PCI master, terminates a transaction (except for a special-cycle) using master-abort. |
| 12 | Received target-abort | Set whenever a PCI transaction initiated by this PCI controller (excluding a special-cycle) is terminated by a target-abort. |
| 11 | Signaled target-abort | Set whenever this PCI controller, acting as the PCI target, issues a target-abort to a PCI master. |
| 10–9 | DEVSEL timing | Hard-wired to 0b00, indicating that this PCI controller uses fast device select timing. |
| 8 | Master data parity error detected | Set upon detecting a data parity error. Three conditions must be met for this bit to be set:<br>• This PCI controller detected a parity error.<br>• This PCI controller was acting as the bus master for the operation in which the error occurred.<br>• Bit 6 in the PCI bus command register was set. |
| 7 | Fast back-to-back capable | Hard-wired to 1, indicating that this PCI controller (as a target) is capable of accepting fast back-to-back transactions. |
| 6 | — | Reserved |
| 5 | 66-MHz capable | Read-only bit indicates that this PCI controller is capable of 66 MHz PCI bus operation. |
| 4 | Capabilities List | Hard-wired to 0 |
| 3–0 | — | Reserved |

### 16.3.2.5 PCI Revision ID Register—Offset 0x08

The PCI revision ID register is used to identify the revision of the part.

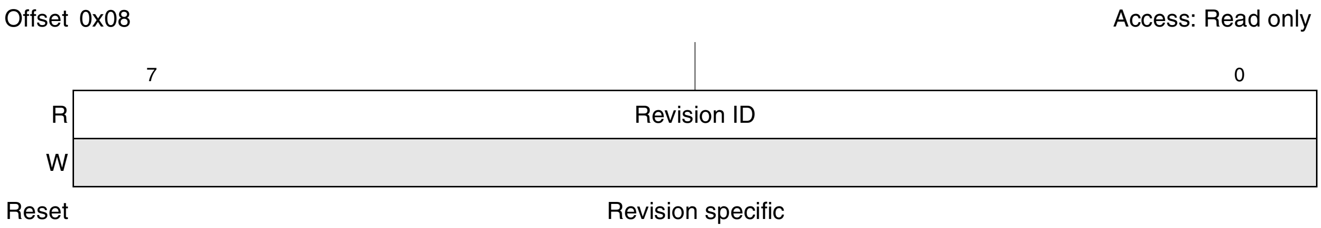


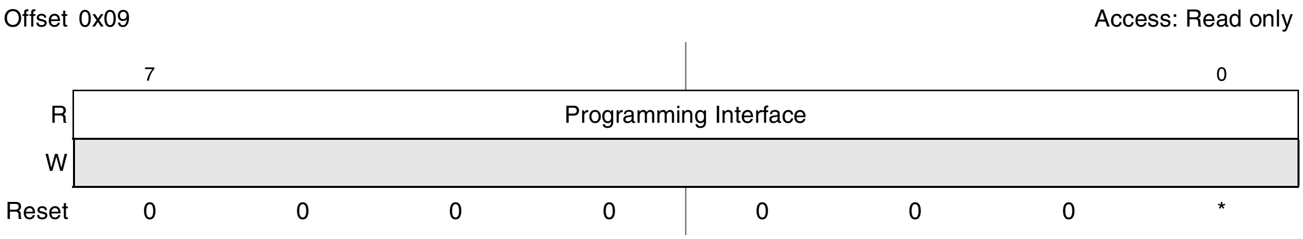
**Figure 16-29. PCI Revision ID Register**

**Table 16-28. PCI Revision ID Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 7–0 | Revision ID | Revision specific |

### 16.3.2.6 PCI Bus Programming Interface Register—Offset 0x09

Table 16-29 describes the PCI bus programming interface register (PIR).



PCI: *0 = Host, 1 = Agent

**Figure 16-30. PCI Bus Programming Interface Register**

**Table 16-29. PCI Bus Programming Interface Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 7–0 | Programming Interface | 0x00 When the PCI controller is configured as host bridge<br>0x01 When the PCI controller is configured as an agent device |

### 16.3.2.7 PCI Subclass Code Register—Offset 0x0A

Table 16-31 describes the PCI subclass code register (PSCR).

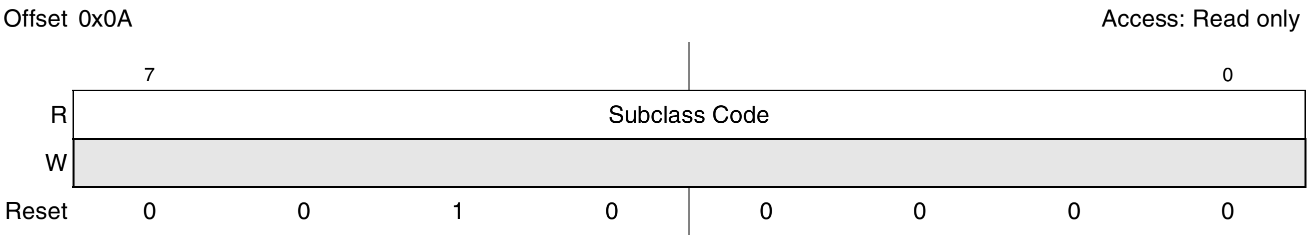


**Figure 16-31. PCI Subclass Code Register**

**Table 16-30. PCI Subclass Code Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 7–0 | Subclass Code | PowerPC—0x20 |

### 16.3.2.8 PCI Bus Base Class Code Register—Offset 0x0B

Table 16-31 describes the PCI bus base class code register (PBCCR).

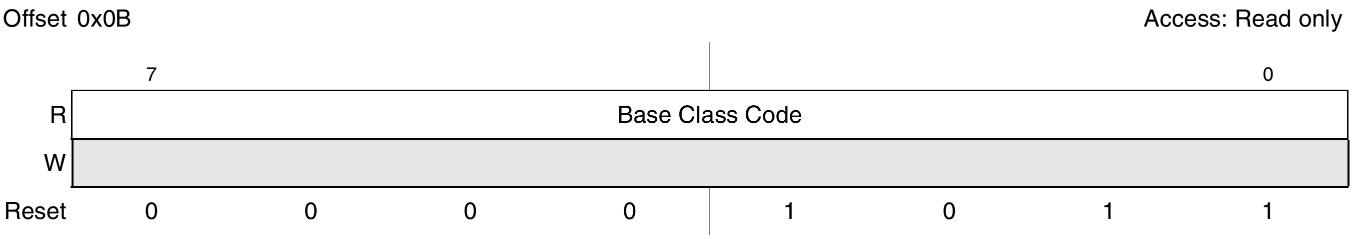


**Figure 16-32. PCI Bus Base Class Code Register**

**Table 16-31. PCI Bus Base Class Code Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 7–0 | Base Class Code | Processor—0x0B |

### 16.3.2.9 PCI Bus Cache Line Size Register—Offset 0x0C

Table 16-32 describes the PCI bus cache line size register (PCLSR).

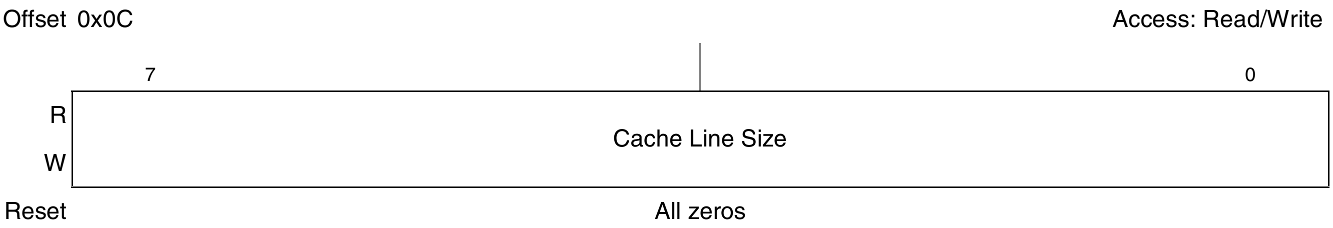


**Figure 16-33. PCI Bus Cache Line Size Register**

**Table 16-32. PCI Bus Cache Line Size Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 7–0 | Cache Line Size | Represents the cache line size of the processor in terms of 32-bit words (8 32-bit words = 32 bytes). PCLSR is read-write; however, for PCI operation an attempt to program this register to any value other than 0x8 results in clearing it. |

### 16.3.2.10 PCI Bus Latency Timer Register—0x0D

Table 16-33 describes the PCI latency timer register (PLTR).

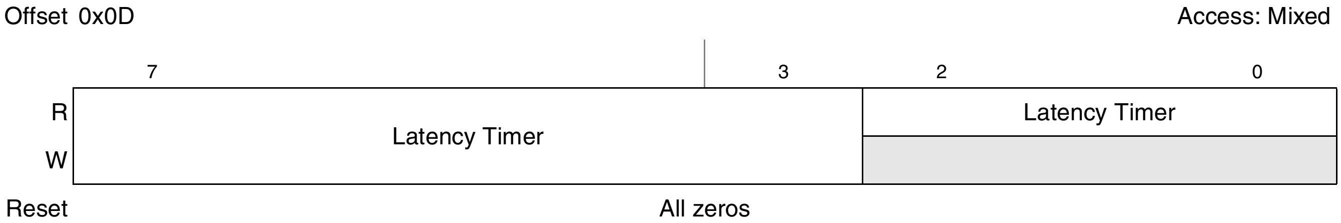


**Figure 16-34. PCI Bus Latency Timer Register**

**Table 16-33. PCI Bus Latency Timer Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 7–3 | Latency Timer | The maximum number of PCI clocks that the device, when mastering a transaction, holds the bus after PCI bus grant has been negated The value is in PCI clocks. The PCI 2.2 specification gives rules by which the PCI bus interface unit completes transactions when the timer has expired. |
| 2–0 | Latency Timer | Read-only bits. The minimum latency timer value when set is 8 PCI clocks. |

### 16.3.2.11 PCI Base Address Registers

A PCI base address register points to the beginnings of each address range to which the device responds by asserting $\overline{\text{PCI\_DEVSEL}}$. The base address register (BAR) at offset 0x10 is a fixed 1-Mbyte window that is automatically translated to the local configuration, control, and status registers address space.

The other base address registers are aliases (with differing format) of the PCI inbound ATMU windows; see Section 16.3.1.3, “PCI ATMU Inbound Registers.” The 32-bit base address register at offset 0x14 corresponds to inbound ATMU window 1; the 64-bit base address registers at offsets 0x18 and 0x20 correspond to inbound ATMU windows 2 and 3. If one of these registers is written, the corresponding ATMU register is also updated; if a PCI inbound ATMU register is written, the corresponding BAR is also updated. If one of these registers is read, the corresponding size of ATMU is returned on the PCI bus providing valid window size in the Inbound ATMU window attributes register.

Note that PCSRBAR cannot be updated through the inbound ATMU registers.

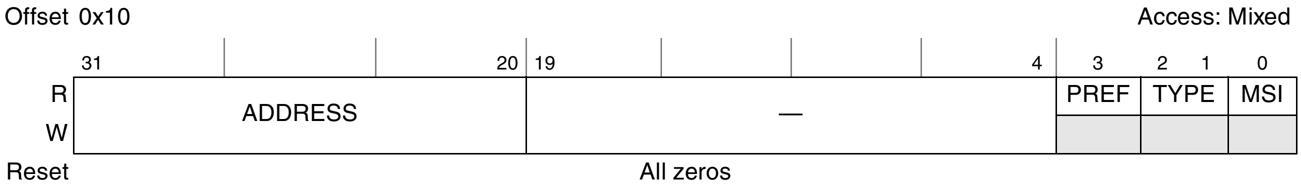


**Figure 16-35. PCI Configuration and Status Register Base Address Register (PCSRBAR)**

**Table 16-34. PCSRBAR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31–20 | ADDRESS | Indicates the base address that the inbound configuration/run-time window resides at. This window is fixed at 1 Mbyte. |
| 19–4 | — | Reserved |
| 3 | PREF | Prefetchable |
| 2–1 | TYPE | Type.<br>00 Locate anywhere in 32-bit address space. |
| 0 | MSI | Memory space indicator |

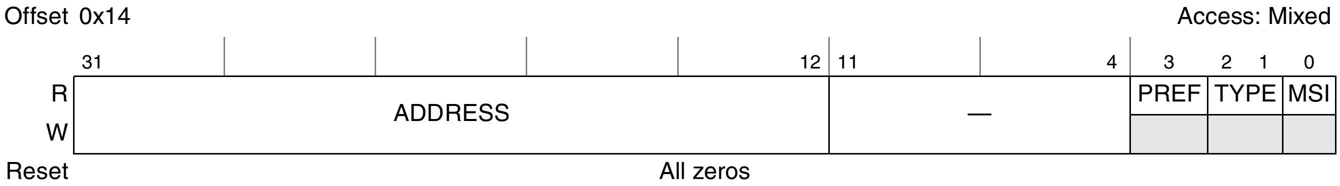


**Figure 16-36. 32-Bit Memory Base Address Register**

**Table 16-35. 32-Bit Memory Base Address Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31–12 | ADDRESS | Indicates the base address that the inbound memory window resides at. The number of upper bits that the device allows to be writable is selected through the inbound translation windows. |
| 11–4 | — | Reserved. The device allows a 4 Kbyte window minimum. |
| 3 | PREF | Prefetchable |
| 2–1 | TYPE | Type.<br>00 Locate anywhere in 32-bit address space. |
| 0 | MSI | Memory space indicator. |

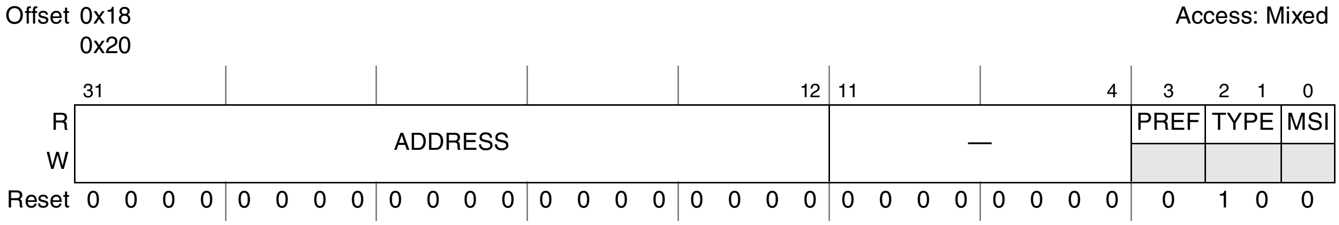


**Figure 16-37. 64-Bit Low Memory Base Address Register**

**Table 16-36. 64-Bit Low Memory Base Address Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31–12 | ADDRESS | Indicates the base address that the inbound memory window resides at. The number of upper bits that the device allows to be writable is selected through the inbound translation windows. |
| 11–4 | — | Reserved. The device allows a 4 Kbyte window minimum. |
| 3 | PREF | Prefetchable |
| 2–1 | TYPE | Type.<br>10 Locate anywhere in 64-bit address space. |
| 0 | MSI | Memory space indicator |

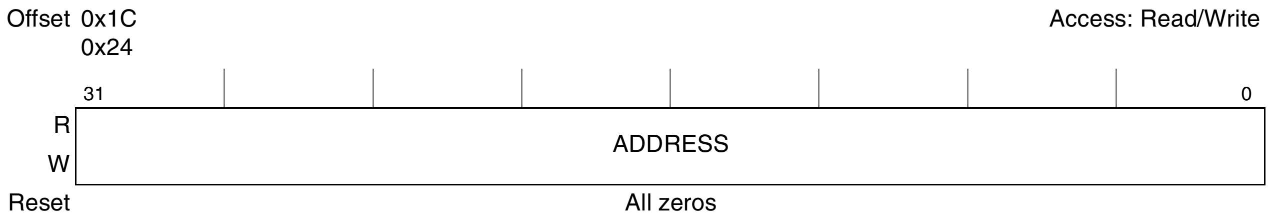


**Figure 16-38. 64-Bit High Memory Base Address Register**

**Table 16-37. Bit Setting for 64-Bit High Memory Base Address Register**

| Bits | Name | Description |
|---|---|---|
| 31–0 | ADDRESS | Indicates the base address that the inbound memory window resides at. The number of upper bits that the device allows to be writable is selected through the inbound translation windows. If no access to local memory is to be permitted by external masters then all bits are programmed. |

## 16.3.2.12 PCI Subsystem Vendor ID Register

The PCI subsystem vendor ID register is used to identify the subsystem.

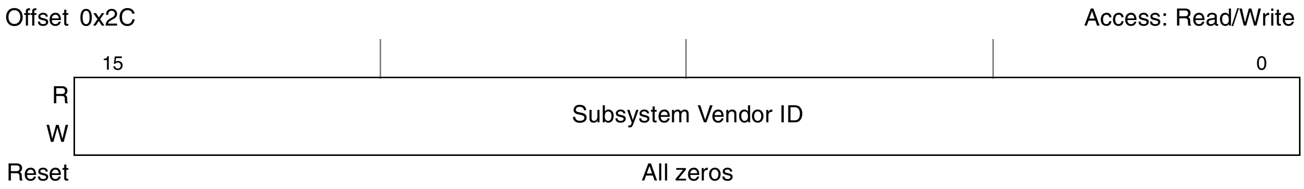


**Figure 16-39. PCI Subsystem Vendor ID Register**

**Table 16-38. PCI Subsystem Vendor ID Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 15–0 | Subsystem Vendor ID | 0x0000 |

### 16.3.2.13 PCI Subsystem ID Register

The PCI subsystem ID register is used to identify the subsystem.

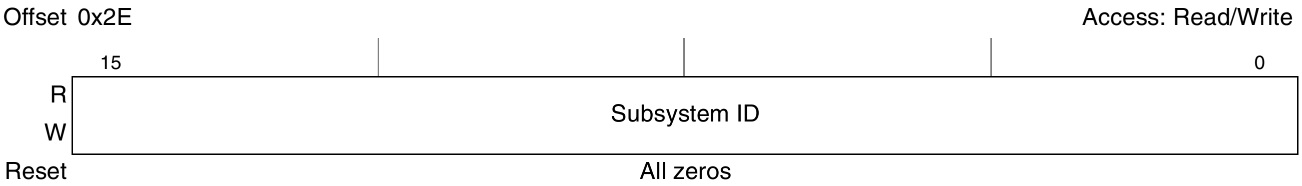


**Figure 16-40. PCI Subsystem ID Register**

**Table 16-39. PCI Subsystem ID Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 15–0 | Subsystem ID | 0x0000 |

### 16.3.2.14 PCI Bus Capabilities Pointer Register

The PCI bus capabilities pointer identifies additional functionality supported by the device.

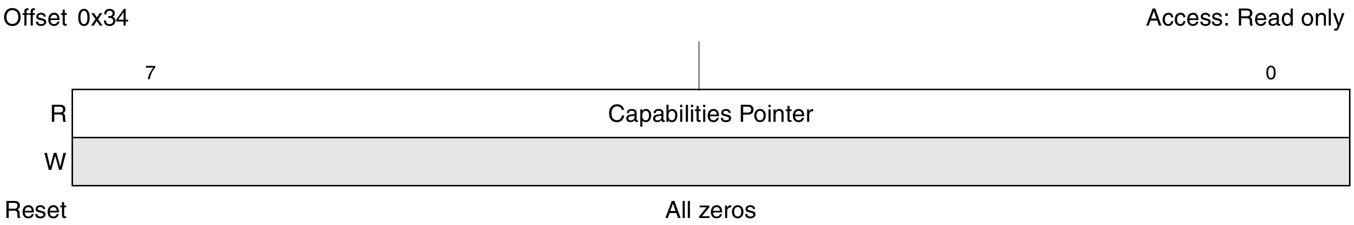


**Figure 16-41. PCI Bus Capabilities Pointer Register**

**Table 16-40. PCI Bus Capabilities Pointer Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 7–0 | Capabilities Pointer | No additional capabilities |

### 16.3.2.15 PCI Bus Interrupt Line Register

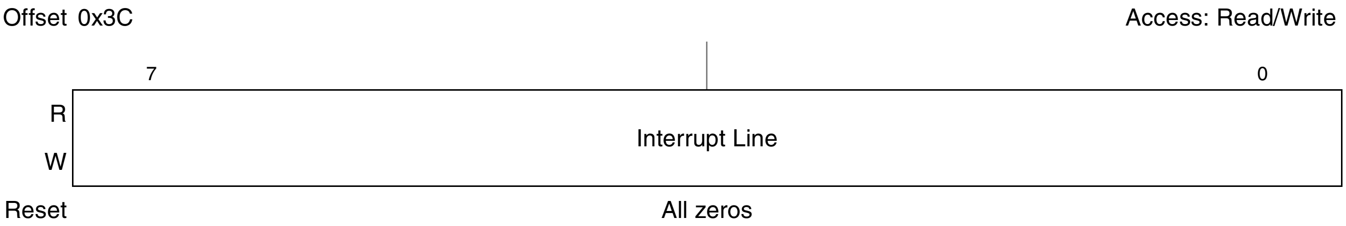


**Figure 16-42. PCI Bus Interrupt Line Register**

**Table 16-41. PCI Bus Interrupt Line Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 7–0 | Interrupt Line | Used to communicate interrupt line routing information. |

### 16.3.2.16 PCI Bus Interrupt Pin Register

The programmable interrupt controller (PIC) has 12 general purpose interrupt request inputs (IRQ[0:11]) and an interrupt output, $\overline{\text{IRQ\_OUT}}$ (active low, level sensitive), to which all external and most internal interrupt sources (including PCI) can be routed. $\overline{\text{IRQ\_OUT}}$ is mapped to PCI INTA# as a default. Note that this device does not respond to INTACK or special cycle commands on the PCI interfaces.

Offset 0x3D Access: Read only

| | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|
| R | Interrupt Pin | | | | | | | |
| W | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

**Figure 16-43. PCI Bus Interrupt Pin Register**

**Table 16-42. PCI Bus Interrupt Pin Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 7–0 | Interrupt pin | PCI_INTA pin selected |

### 16.3.2.17 PCI Bus Minimum Grant Register (MIN_GNT)

Offset 0x3E Access: Read only

| | 7 … 0 |
|---|---|
| R | MINGNT |
| W | |
| Reset | All zeros |

**Figure 16-44. PCI Bus Minimum Grant Register (MIN_GNT)**

**Table 16-43. PCI Bus Minimum Grant Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 7–0 | MINGNT | Specifies the length of the device’s burst period (0x00 indicates that this PCI controller has no major requirements for the settings of latency timers.) |

### 16.3.2.18 PCI Bus Maximum Latency Register (MAX_LAT)

Offset 0x3F

Access: Read Only

| | 7 … 0 |
|---|---|
| R | MAXLAT |
| W | |
| Reset | All zeros |

**Figure 16-45. PCI Bus Maximum Latency Register (MAX_LAT)**

**Table 16-44. PCI Bus Maximum Latency Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 7–0 | MAXLAT | Specifies how often the device needs to gain access to the PCI bus (0x00 indicates that this PCI controller has no major requirements for the settings of latency timers.) |

### 16.3.2.19 PCI Bus Function Register (PBFR)

The 2-byte PCI bus function register is used to determine how different features of the PCI interface in bus 0 are configured. This register is in PCI configuration space at offset 0x44.

Offset 0x44

Access: Mixed

| | 15 | | | | | | | | | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | — | | | | | | | | | | ACL | — | | | | PAH |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | * | 0 | 0 | 0 | 0 | * |

* = Depends on the state of the reset configuration signals at reset

**Figure 16-46. PCI Bus Function Register**

**Table 16-45. PCI Bus Function Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 15–6 | — | Reserved |
| 5 | ACL | Agent configuration lock. Indicates to an external host whether the local processor is doing internal configuration and must be explicitly set and cleared by the local processor during this time. ACL is set during reset if the cfg_cpu_boot configuration input selects the CPU as the configuration owner. (See Section 4.4.3.10, “CPU Boot Configuration.”) This bit is only meaningful in agent mode.<br>0 PCI interface allows incoming PCI configuration cycles.<br>1 PCI interface retries all incoming PCI configuration cycles. |
| 4–1 | — | Reserved |
| 0 | PAH | PCI agent/host. Read-only. Indicates the reset value of the cfg_host_agt configuration signal.<br>0 PCI interface is in host mode<br>1 PCI interface is in agent mode |

### 16.3.2.20 PCI Bus Arbiter Configuration Register (PBACR)

The PCI bus arbiter configuration register is used to determine the configuration of the PCI bus arbiter.

Offset 0x46 Access: Mixed

| | 15 | 14 | 13 | 12 | 11 … 7 | 6 … 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| R | PAD | PM | — | PBMD | — | PBMP | — | DP |
| W | | | | | | | | |
| Reset | All zeros | | | | | | | |

**Figure 16-47. PCI Bus Arbiter Configuration Register**

**Table 16-46. PCI Bus Arbiter Configuration Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 15 | PAD | PCI arbiter disable. Determines if the device is the PCI arbiter on the PCI bus or not. The reset state is determined by the inverse of the cfg_pci*n*_arb configuration input signal when reset is released.<br>0 Device is the PCI arbiter.<br>1 Device is not the PCI arbiter. Device presents its request on $\overline{\text{PCI\_REQ0}}$ to the external arbiter and receives its grant on $\overline{\text{PCI\_GNT0}}$. |
| 14 | PM | Parking mode. controls which device receives the bus grant when there are no outstanding bus requests and the bus is idle.<br>0 The bus is parked on the last device to use the bus.<br>1 The bus is parked on the device. |
| 13 | — | Reserved |
| 12 | PBMD | PCI broken master disable. Determines if the device ignores the bus requests of an initiator that requests the bus for an excessive period without using the bus.<br>0 An initiator that requests the bus and receives the grant must begin using the bus within 16 PCI clock periods after the bus becomes idle or else its request is subsequently ignored.<br>1 No requests are ignored. |
| 11–7 | — | Reserved |
| 6–2 | PBMP | PCI bus master priorities. Determines arbitration priority given to different masters on the PCI bus. Bit 6 corresponds to the priority of the master sourcing $\overline{\text{PCI\_REQ0}}$; bit 2 corresponds to the priority of the master sourcing $\overline{\text{PCI\_REQ4}}$.<br>0 Master *n* is low priority.<br>1 Master *n* is high priority. |
| 1 | — | Reserved |
| 0 | DP | Device priority. Determines this device’s arbitration priority.<br>0 Device is low priority.<br>1 Device is high priority. |

## 16.4 Functional Description

This section describes the functionality of the PCI interface.

### 16.4.1 PCI Bus Arbitration

PCI bus arbitration is access-based. Bus masters must arbitrate for each access performed on the bus. The PCI bus uses a central arbitration scheme where each master has its own unique request ($\overline{\text{REQ}}$) output and grant ($\overline{\text{GNT}}$) input signal. A simple request/grant handshake is used to gain access to the bus. Arbitration for the bus occurs during the previous access so that no PCI bus cycles are consumed due to arbitration (except when the bus is idle).

The PCI controller provides bus arbitration logic for its master interface and up to five other external PCI bus masters. The on-chip PCI arbiter is independent of host or agent mode. The on-chip PCI arbiter functions in both host and agent modes, or it can be disabled to allow for an external PCI arbiter.

A configuration signal sampled at the negation of the reset signal (HRESET) determines if the on-chip PCI arbiter is enabled (high) or disabled (low). The on-chip PCI arbiter can also be enabled or disabled by programming bit 15 of the PCI bus arbitration control register (PBACR[PAD]). Note that the sense of PBACR[PAD] corresponds to the inverse of the configuration signal (that is, when PAD = 0 the arbiter is enabled, and when PA = 1 the arbiter is disabled). See Chapter 4, "Reset, Clocking, and Initialization," for more information on the reset configuration signals.

If the on-chip PCI arbiter is enabled, a request-grant pair of signals is provided for each external master (PCI_REQ[0:4] and PCI_GNT[0:4]). In addition, there is an internal request/grant pair for the internal master state machine that governs internal accesses to the PCI interface. If the on-chip PCI arbiter is disabled, the PCI controller uses the PCI_REQ0 signal as an output to issue its request to the external arbiter and uses the PCI_GNT0 signal as an input to receive its grant from the external arbiter.

The following sections describe the operation of the on-chip PCI arbiter that arbitrates between external PCI masters and the internal PCI bus master.

#### 16.4.1.1 PCI Bus Arbiter Operation

The on-chip PCI arbiter uses a programmable two-level, round-robin arbitration algorithm. Each of the five external masters, plus the device itself, can be programmed for two priority levels, high or low, using the appropriate bits in the PBACR. Within each priority group, the PCI bus grant is asserted to the next requesting device in numerical order, with the PCI controller positioned before device 0.

Conceptually, the lowest priority device is the master that is currently using the bus, and the highest priority device is the device that follows the current master in numerical order and group priority. This is considered to be a fair algorithm, since a single device cannot prevent other devices from having access to the bus; it automatically becomes the lowest priority device as soon as it begins to use the bus. If a master is not requesting the bus, then its transaction slot is given to the next requesting device within its priority group.

A grant is awarded to the highest priority requesting device as soon as the current master begins a transaction; however, the granted device must wait until the bus is relinquished by the current master before initiating a transaction.

The grant given to a particular device may be removed and awarded to another higher priority device, whenever the higher priority device asserts its request. If the bus is idle when a device requests the bus, then the arbiter withholds the grant for one clock cycle. The arbiter re-evaluates the priorities of all requesting devices and grants the bus to the highest priority device in the following clock cycle. This allows a turnaround clock when a higher priority device is using address stepping or when the bus is parked.

The low-priority group collectively has one bus transaction request slot in the high-priority group. For N high-priority devices and M low-priority devices, each high-priority device is guaranteed at least 1 of N+1 bus transactions and each low-priority device is guaranteed at least 1 of $(N+1) \times M$ bus transactions, with one low-priority device receiving the grant in 1 of N+1 bus transactions. If all devices are programmed to the same priority level, or if the low-priority group has only one device, the algorithm defaults to give each device an equal number of bus grants in round-robin sequence.

For the example in Figure 16-48, assume that several devices are requesting the bus. If two masters are in the high-priority group and three are in the low-priority group, each high-priority master is guaranteed at least one out of three transaction slots and each low-priority master is guaranteed one out of nine transaction slots.

In Figure 16-48, the grant sequence (with all devices, except device 4 requesting the bus and device 3 being the current master) is 0, 2, MPC8536E, 0, 2, 1, 0, 2, 3, …, and repeating. If device 2 is not requesting the bus, the grant sequence is 0, MPC8536E, 0, 1, 0, 3, …, and repeating. If device 2 requests the bus when device 0 is conducting a transaction and the MPC8536E has the next grant, the MPC8536E has its grant removed and device 2 is awarded the grant since device 2 is higher priority than the MPC8536E when device 0 has the bus.

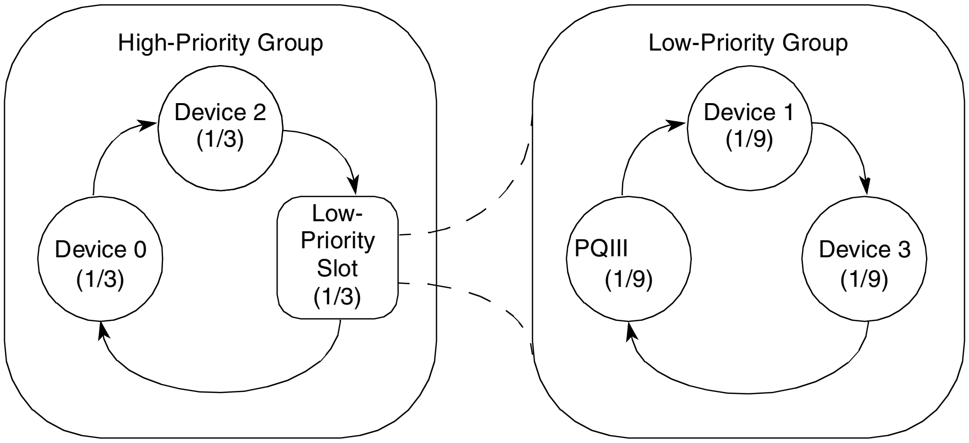


**Figure 16-48. PCI Arbitration Example**

### 16.4.1.2 PCI Bus Parking

When no device is using or requesting the bus, the PCI arbiter grants the bus to a selected device. This is known as parking the bus on the selected device. The selected device is required to drive the PCI_AD[31:0], PCI_C/$\overline{\text{BE}}$[0:3], and the PCI parity signals to a stable value, preventing these signals from floating.

The parking mode parameter (PBACR[PM]) determines which device the arbiter selects for parking the PCI bus. If PBACR[PM] = 0 (or if the bus is not idle), then the bus is parked on the last master to use the bus. If the bus is idle and PBACR[PM] = 1, the bus is parked on the PCI controller.

### 16.4.1.3 Broken Master Lock-Out

The PCI bus arbiter has a feature that allows it to lock out any masters that are broken or ill-behaved. The broken master feature is controlled by programming bit 12 of the PCI bus arbitration control register (0 = enabled, 1 = disabled).

When the broken master feature is enabled, a granted device that does not assert $\overline{\text{PCI\_FRAME}}$ within 16 PCI clock cycles after the bus is idle, has its grant removed and subsequent requests are ignored until its

REQ is negated for at least one clock cycle. This prevents ill-behaved masters from monopolizing the bus. When the broken master feature is disabled, a device that requests the bus and receives a grant never loses its grant until and unless it begins a transaction or negates its REQ signal. Note that disabling the broken master feature is not recommended.

#### 16.4.1.4 Power-Saving Modes and the PCI Arbiter

In the sleep power-saving mode, the clock signal driving SYSCLK can be disabled. If the clock is disabled, the arbitration logic is not able to perform its function. System programmers must park the bus with a device that can sustain the PCI_AD[31:0], PCI_C/BE[3:0], and parity signals prior to disabling the SYSCLK signal. If the bus is parked on the MPC8536E when its clocks are stopped, the MPC8536E sustains the PCI_AD[31:0], PCI_C/BE[3:0], and parity signals in their prior states. In this situation, the only way for another agent to use the PCI bus is by waking the MPC8536E. In nap and doze power-saving modes, the arbiter continues to operate allowing other PCI devices to run transactions.

### 16.4.2 PCI Bus Protocol

This section provides a general description of the PCI bus protocol. Specific PCI bus transactions are described in Section 16.4.2.7, “PCI Bus Transactions.” Refer to Figure 16-49, Figure 16-50, Figure 16-51, and Figure 16-52 for examples of the transfer-control mechanisms described in this section.

All signals are sampled on the rising edge of the PCI bus clock (SYSCLK). Each signal has a setup and hold aperture with respect to the rising clock edge in which transitions are not allowed. Outside this aperture, signal values or transitions have no significance. See the separate hardware specifications document for specific setup and hold times.

#### 16.4.2.1 Basic Transfer Control

The basic PCI bus transfer mechanism is a burst. A burst is composed of an address phase followed by one or more data phases. Fundamentally, all PCI data transfers are controlled by three signals—PCI_FRAME (frame), PCI_IRDY (initiator ready), and PCI_TRDY (target ready). An initiator asserts PCI_FRAME to indicate the beginning of a PCI bus transaction and negates PCI_FRAME to indicate the end of a PCI bus transaction. An initiator negates PCI_IRDY to force wait cycles. A target negates PCI_TRDY to force wait cycles.

The PCI bus is considered idle when both PCI_FRAME and PCI_IRDY are negated. The first clock cycle in which PCI_FRAME is asserted indicates the beginning of the address phase. The address and bus command code are transferred in that first cycle. The next cycle begins the first of one or more data phases. Data is transferred between initiator and target in each cycle that both PCI_IRDY and PCI_TRDY are asserted. Wait cycles may be inserted in a data phase by the initiator (by negating PCI_IRDY) or by the target (by negating PCI_TRDY).

Once an initiator has asserted PCI_IRDY, it cannot change PCI_IRDY or PCI_FRAME until the current data phase completes regardless of the state of PCI_TRDY. Once a target has asserted PCI_TRDY or PCI_STOP, it cannot change PCI_DEVSEL, PCI_TRDY, or PCI_STOP until the current data phase completes. In simpler terms, once an initiator or target has committed to the data transfer, it cannot change its mind.

When the initiator intends to complete only one more data transfer (which could be immediately after the address phase), PCI_FRAME is negated and PCI_IRDY is asserted (or kept asserted), indicating the initiator is ready. After the target indicates the final data transfer (by asserting PCI_TRDY), the PCI bus may return to the idle state (both PCI_FRAME and PCI_IRDY are negated) unless a fast back-to-back transaction is in progress. In the case of a fast back-to-back transaction, an address phase immediately follows the last data phase.

### 16.4.2.2 PCI Bus Commands

A PCI bus command is encoded in the PCI_C/BE[3:0] signals during the address phase of a PCI transaction. The bus command indicates to the target the type of transaction the initiator is requesting. Table 16-47 describes the PCI bus commands implemented by the device.

**Table 16-47. PCI Bus Commands**

| PCI_C/ BE[3:0] | PCI Bus Command | Supported as an Initiator | Supported as a Target | Definition |
|---|---|---|---|---|
| 0000 | Interrupt-acknowledge | Yes | No | A read (implicitly addressing the system interrupt controller). Only one device on the PCI bus should respond to this command; others ignore it. See Section 16.4.2.12.1, “Interrupt-Acknowledge Transactions,” for more information. |
| 0001 | Special cycle | Yes | No | Provides a way to broadcast select messages to all devices on the PCI bus. See Section 16.4.2.12.2, “Special-Cycle Transactions,” for more information. |
| 0010 | I/O-read | Yes | No | Accesses agents mapped into the PCI I/O space. |
| 0011 | I/O-write | Yes | No | Accesses agents mapped into the PCI I/O space. |
| 0100 | Reserved[1] | No | No | — |
| 0101 | Reserved[1] | No | No | — |
| 0110 | Memory-read | Yes | Yes | Accesses either local memory or agents mapped into PCI memory space, depending on the address. When a PCI master issues this command to local memory, the PCI controller (the target) fetches data from the requested address to the end of the cache line (32 bytes) from local memory, even though all of the data may not be requested by (or sent to) the initiator. |
| 0111 | Memory-write | Yes | Yes | Accesses either local memory or agents mapped into PCI memory space, depending on the address. |
| 1000 | Reserved[1] | No | No | — |
| 1001 | Reserved[1] | No | No | — |
| 1010 | Configuration-read | Yes | Agent mode only | Accesses the 256-byte configuration space of a PCI agent. A specific agent is selected when its IDSEL signal is asserted during the address phase. See Section 16.4.2.11, “Configuration Cycles,” for details. |
| 1011 | Configuration-write | Yes | Agent mode only | |
| 1100 | Memory-read-multiple | Yes | Yes | Similar to the memory-read command, but also causes a prefetch of the next cache line (32 bytes). |
| 1101 | Dual-address-cycle | Yes | Yes | Used to transfer a 64-bit address (in two 32-bit address cycles) to 64-bit addressable devices. |

Table 16-47. PCI Bus Commands (continued)

| PCI_C/<br>BE[3:0] | PCI Bus Command | Supported as an Initiator | Supported as a Target | Definition |
|---|---|---|---|---|
| 1110 | Memory-read-line | Yes | Yes | Indicates that an initiator is requesting the transfer of an entire cache line. This occurs only when the processor is performing a burst read. Note that these processors perform burst reads only when the appropriate cache is enabled and the transaction is not cache-inhibited. |
| 1111 | Memory-write-and-invalidate | No | Yes | Indicates that an initiator is transferring an entire cache line; if this data is in any cacheable memory, that cache line needs to be invalidated. |

[1] Reserved command encodings are reserved for future use. The PCI controller does not respond to these commands.

### 16.4.2.3 Addressing

PCI defines three physical address spaces—PCI memory space, PCI I/O space, and PCI configuration space. Access to the PCI memory and I/O space is straightforward, although one must take into account the local memory access window and address translation being used. The address translation registers are described in Section 16.3.1, “PCI Memory-Mapped Registers.” Access to the PCI configuration space is described in Section 16.4.2.11, “Configuration Cycles.”

Address decoding on the PCI bus is performed by every device for every PCI transaction. Each agent is responsible for decoding its own address. PCI supports two types of address decoding—positive decoding and subtractive decoding. For positive decoding, each device looks for accesses in the address range that the device has been assigned. For subtractive decoding, one device on the bus looks for accesses that no other device has claimed. See Section 16.4.2.4, “Device Selection,” for information about claiming transactions.

The information contained in the two low-order address bits (PCI_AD[1:0]) varies by the address space (memory, I/O, or configuration). Regardless of the encoding scheme, the two low-order address bits are always included in parity calculations.

#### 16.4.2.3.1 Memory Space Addressing

For memory accesses, PCI defines two types of burst ordering controlled by the two low-order bits of the address—linear incrementing (PCI_AD[1:0] = 0b00) and cache wrap mode (PCI_AD[1:0] = 0b10), as shown in Table 16-48. The other two PCI_AD[1:0] possibilities (0b01 and 0b11) are reserved. As an initiator, the PCI controller always encodes PCI_AD[1:0] = 00 for PCI memory space accesses. As a target, the PCI controller executes a target disconnect after the first data phase completes if PCI_AD[1:0] = 01 or PCI_AD[1:0] = 0b11 during the address phase of a local memory access. See Section 16.4.2.8.2, “Target-Initiated Termination,” for more information on target disconnect conditions.

Table 16-48. Supported Combinations of PCI_AD[1:0]

| PCI_AD[1:0] | | Target | | Initiator | |
|---|---|---|---|---|---|
| | | Read | Write | Read | Write |
| 00 | Linear | √ | √ | √ | √ |
| 01 | Reserved | TD | TD | — | — |

Table 16-48. Supported Combinations of PCI_AD[1:0] (continued)

| PCI_AD[1:0] | | Target | | Initiator | |
|---|---|---|---|---|---|
| | | Read | Write | Read | Write |
| 10 | Cache Wrap | √ | TD | — | — |
| 11 | Reserved | TD | TD | — | — |

For linear incrementing mode, the memory address is encoded/decoded using PCI_AD[31:2]. Thereafter, the address is incremented by 4 bytes after each data phase completes until the transaction is terminated or completed (a 4-byte data width per data phase is implied). Note that the two low-order bits on the address bus are included in all parity calculations.

For cache wrap mode (PCI_AD[1:0] = 0b10) reads, the critical memory address is decoded using PCI_AD[31:2]. The address is incremented by 4 bytes after each data phase completes until the end of the cache line is reached. For cache-wrap reads, the address wraps to the beginning of the current cache line and continues incrementing until the entire cache line (32 bytes) is read. The PCI controller does not support cache-wrap write operations and executes a target disconnect after the data phase for the end of the cache line completes for writes with PCI_AD[1:0] = 0b10. That is, the PCI controller does not wrap back to the beginning of the cache line. Note that the two low-order bits on the address bus are included in all parity calculations.

#### 16.4.2.3.2 I/O Space Addressing

For PCI I/O accesses, 32 address signals (PCI_AD[31:0]) are used to provide a byte address. After a target has claimed an I/O access, it must determine if it can complete the entire access as indicated by the byte enable signals. If all the selected bytes are not in the address range of the target, the entire access cannot complete. In this case, the target does not transfer any data and terminates the transaction with a target-abort error. See Section 16.4.2.8.2, “Target-Initiated Termination,” for more information.

#### 16.4.2.3.3 Configuration Space Addressing

PCI supports two types of configuration accesses that use different formats for the PCI_AD[31:0] signals during the address phase. The two low-order bits of the address indicate the format used for the configuration address phase—type 0 (PCI_AD[1:0] = 0b00) or type 1 (PCI_AD[1:0] = 0b01). Both address formats identify a specific device and a specific configuration register for that device. See Section 16.4.2.11, “Configuration Cycles,” for descriptions of the two formats.

### 16.4.2.4 Device Selection

The PCI_DEVSEL signal is driven by the target of the current transaction. PCI_DEVSEL indicates to the other devices on the PCI bus that the target has decoded the address and claimed the transaction. PCI_DEVSEL may be driven one, two, or three clock cycles (fast, medium, or slow device select timing) following the address phase. Device select timing is encoded into the device’s PCI bus status register. If no agent asserts PCI_DEVSEL within three clock cycles of PCI_FRAME, the agent responsible for subtractive decoding may claim the transaction by asserting PCI_DEVSEL.

A target must assert PCI_DEVSEL (claim the transaction) before or coincident with any other target response (assert PCI_TRDY, PCI_STOP, or data signals). In all cases except target-abort, once a target asserts PCI_DEVSEL, it must not negate PCI_DEVSEL until PCI_FRAME is negated (with PCI_IRDY asserted) and the last data phase has completed. For normal termination, negation of PCI_DEVSEL coincides with the negation of PCI_TRDY or PCI_STOP.

If the first access maps into a target's address range, that target asserts PCI_DEVSEL to claim the access. However, if the initiator attempts to continue the burst access across the resource boundary, then the target must issue a target disconnect.

The PCI controller is hardwired for fast device select timing (PCI bus status register [10–9] = 0b00). Therefore, when the PCI controller is the target of a transaction (local memory access or configuration register access), it asserts PCI_DEVSEL one clock cycle following the address phase.

As an initiator, if the PCI controller does not detect the assertion of PCI_DEVSEL within four clock cycles after the address phase (that is, five clock cycles after it asserts PCI_FRAME), it terminates the transaction with a master-abort termination; see Section 16.4.2.8.1, "Master-Initiated Termination."

### 16.4.2.5 Byte Alignment

The byte enable signals of the PCI bus (PCI_C/BE[3:0], during a data phase) are used to determine which byte lanes carry meaningful data. The byte enable signals may enable different bytes for each of the data phases. The byte enables are valid on the edge of the clock that starts each data phase and stay valid for the entire data phase. Note that parity is calculated for all bytes regardless of the state of the byte enable signals. See Section 16.4.2.13.1, "PCI Parity," for more information.

If the PCI controller, as a target, detects no byte enables asserted, it completes the current data phase with no permanent change. This implies that on a read transaction, the PCI controller expects that the data is not changed, and on a write transaction, the data is not stored.

### 16.4.2.6 Bus Driving and Turnaround

To avoid contention, a turnaround cycle is required on all signals that may be driven by more than one agent. The turnaround cycle occurs at different times for different signals. The PCI_IRDY, PCI_TRDY, PCI_DEVSEL, and PCI_STOP signals use the address phase as their turnaround cycle. PCI_FRAME,PCI_C/BE[3:0], and PCI_AD[31:0] signals use the idle cycle between transactions (when both PCI_FRAME and PCI_IRDY are negated) as their turnaround cycle. PCI_PERR has a turnaround cycle on the fourth clock cycle after the last data phase.

The PCI address/data signals, PCI_AD[31:0], are driven to a stable condition during every address/data phase. Even when the byte enables indicate that byte lanes carry meaningless data, the signals carry stable values. Parity is calculated on all bytes regardless of the byte enables. See Section 16.4.2.13.1, "PCI Parity," for more information.

### 16.4.2.7 PCI Bus Transactions

This section provides descriptions of the PCI bus transactions. All bus transactions follow the protocol as described in Section 16.4.2, “PCI Bus Protocol.” Read and write transactions are similar for the memory and I/O spaces, so they are described as generic read transactions and generic write transactions.

The timing diagrams in this section show the relationship of significant signals involved in bus transactions. When a signal is drawn as a solid line, it is actively being driven by the current master or target. When a signal is drawn as a dashed line, no agent is actively driving it. High-impedance signals are indicated to have indeterminate values when the dashed line is between the two rails.

The terms ‘edge’ and ‘clock edge’ always refer to the rising edge of the clock. The terms ‘asserted’ and ‘negated’ always refer to the globally visible state of the signal on the clock edge, and not to signal transitions. ‘ ↻ ’ represents a turnaround cycle in the timing diagrams.

#### 16.4.2.7.1 PCI Read Transactions

This section describes PCI single-beat read transactions and PCI burst read transactions.

A read transaction starts with the address phase, occurring when an initiator asserts $\overline{\text{PCI\_FRAME}}$. During the address phase, PCI_AD[31:0] contains a valid address and PCI_C/$\overline{\text{BE}}$[3:0] contains a valid bus command.

The first data phase of a read transaction requires a turnaround cycle. This allows the transition from the initiator driving PCI_AD[31:0] as address signals to the target driving PCI_AD[31:0] as data signals. The turnaround cycle is enforced by the target with the $\overline{\text{TRDY}}$ signal. The target provides valid data at the earliest one cycle after the turnaround cycle. The target must drive the PCI_AD[31:0] signals when $\overline{\text{PCI\_DEVSEL}}$ is asserted.

During the data phase, the PCI_C/$\overline{\text{BE}}$[3:0] signals indicate which byte lanes are involved in the current data phase. A data phase may consist of a data transfer and wait cycles. The PCI_C/$\overline{\text{BE}}$[3:0] signals remain actively driven for both reads and writes from the first clock of the data phase through the end of the transaction.

A data phase completes when data is transferred, which occurs when both $\overline{\text{PCI\_IRDY}}$ and $\overline{\text{PCI\_TRDY}}$ are asserted on the same clock edge. When either $\overline{\text{PCI\_IRDY}}$ or $\overline{\text{PCI\_TRDY}}$ is negated, a wait cycle is inserted and no data is transferred. The initiator indicates the last data phase by negating $\overline{\text{PCI\_FRAME}}$ when $\overline{\text{PCI\_IRDY}}$ is asserted. The transaction is considered complete when data is transferred in the last data phase.

Figure 16-49 illustrates a PCI single-beat read transaction.

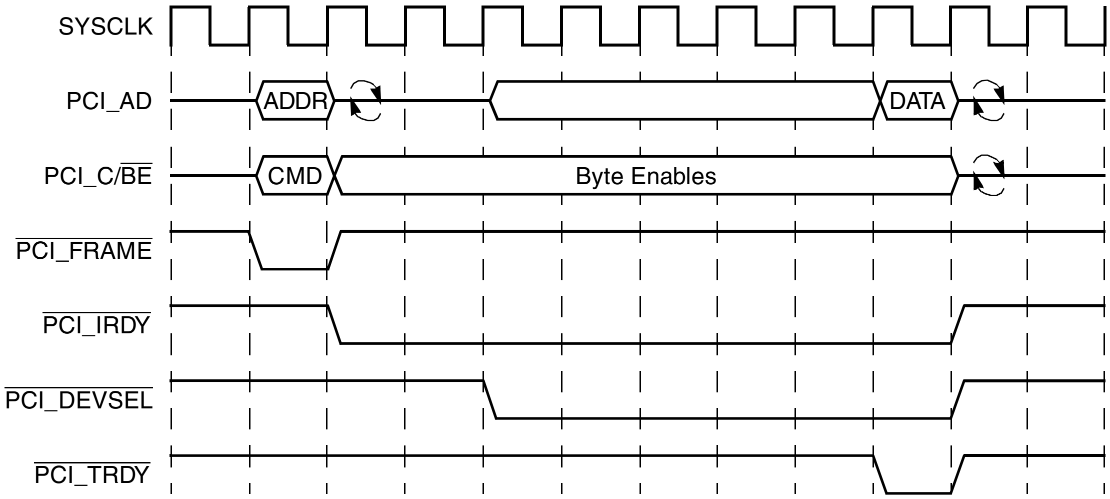


**Figure 16-49. PCI Single-Beat Read Transaction**

Figure 16-50 illustrates a PCI burst read transaction.

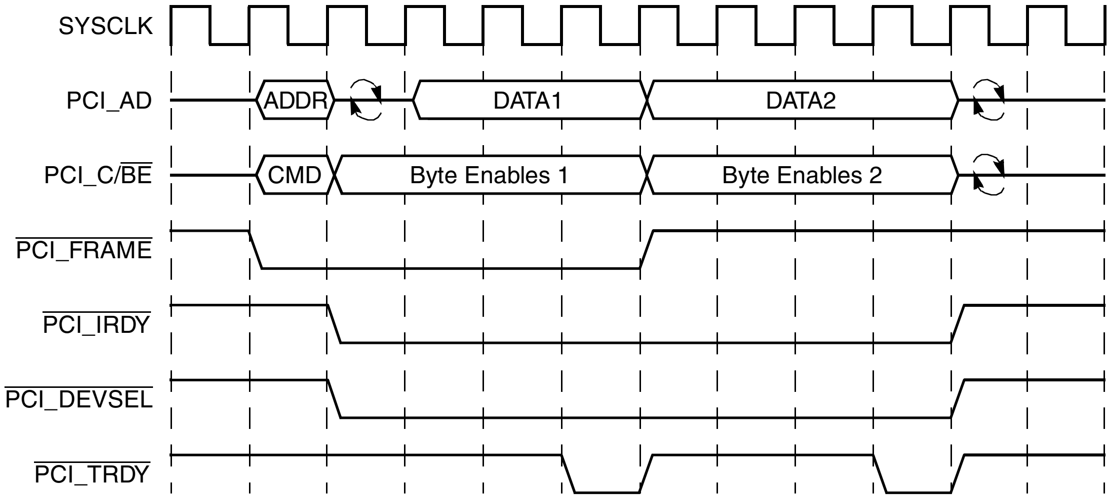


**Figure 16-50. PCI Burst Read Transaction**

### 16.4.2.7.2 PCI Write Transactions

This section describes PCI single-beat write transactions, and PCI burst write transactions. A PCI write transaction starts with the address phase, occurring when an initiator asserts PCI_FRAME. A write transaction is similar to a read transaction except no turnaround cycle is needed following the address phase because the initiator provides both address and data. The data phases are the same for both read and write transactions. Although not shown in the figures, the initiator must drive the PCI_C/BE[3:0] signals, even if the initiator is not ready to provide valid data (PCI_IRDY negated).

Figure 16-51 illustrates a PCI single-beat write transaction.

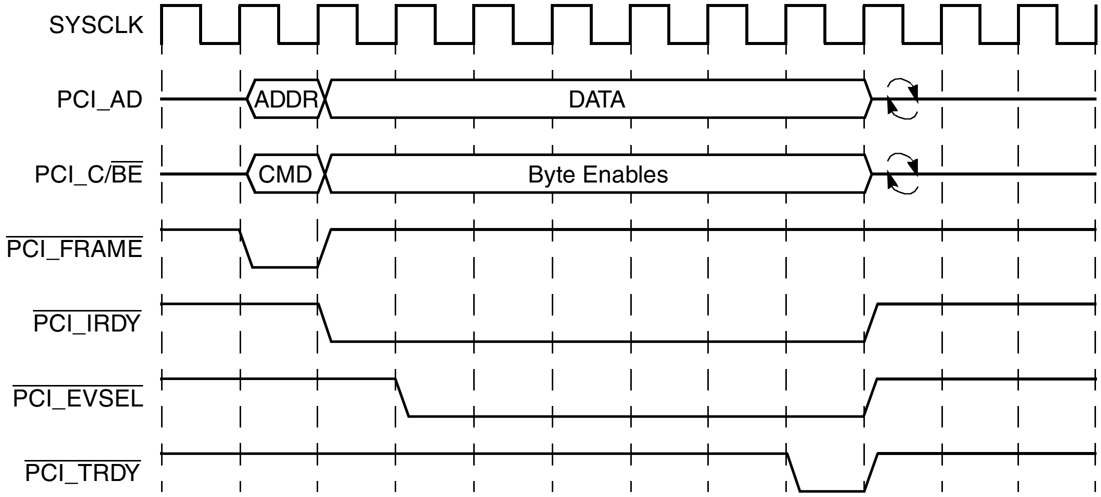


**Figure 16-51. PCI Single-Beat Write Transaction**

Figure 16-52 illustrates a PCI burst write transaction.

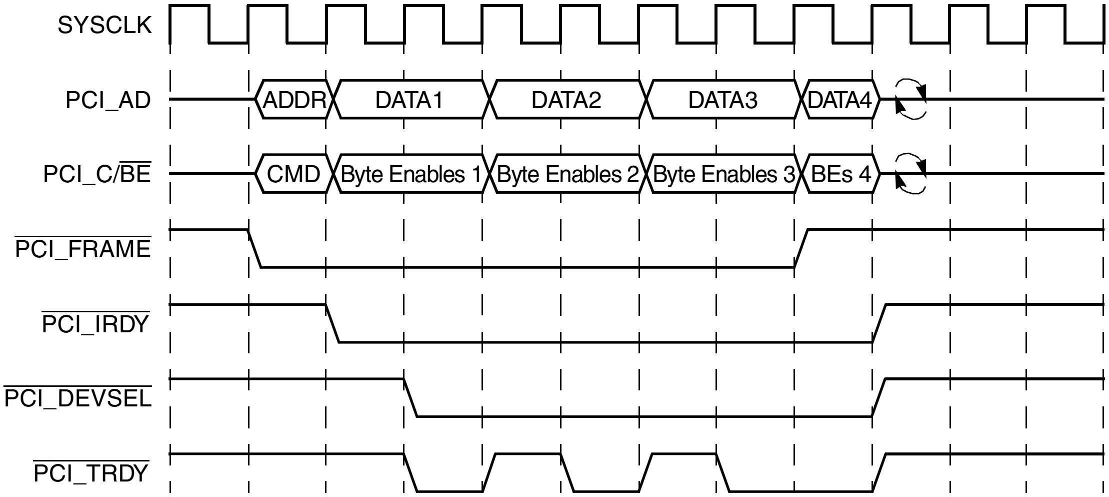


**Figure 16-52. PCI Burst Write Transaction**

### 16.4.2.8 Transaction Termination

A PCI transaction may be terminated by either the initiator or the target. The initiator is ultimately responsible for concluding all transactions, regardless of the cause of the termination. All transactions are concluded when PCI_FRAME and PCI_IRDY are both negated, indicating the bus is idle.

#### 16.4.2.8.1 Master-Initiated Termination

Normally, a master initiates termination by negating PCI_FRAME and asserting PCI_IRDY. This indicates to the target that the final data phase is in progress. The final data transfer occurs when both PCI_TRDY and PCI_IRDY are asserted. The transaction is considered complete when data is transferred in the last data phase. After the final data phase, both PCI_FRAME and PCI_IRDY are negated (the bus becomes idle).

There are three types of master-initiated termination:

- Completion—Refers to termination when the initiator has concluded its intended transaction. This is the most common reason for termination.
- Timeout—Refers to termination when the initiator loses its bus grant (GNT*n* is negated), and its internal latency timer has expired. The intended transaction is not necessarily concluded.
- Master-abort—An abnormal case of master-initiated termination. If no device (including the subtractive decoding agent) asserts PCI_DEVSEL to claim a transaction, the initiator terminates the transaction with a master-abort. For a master-abort termination, the initiator negates PCI_FRAME and then negates PCI_IRDY on the next clock. If a transaction is terminated by master-abort (except for a special-cycle command), the received master-abort bit (bit 13) of the PCI bus status register is set.

As an initiator, if the PCI controller does not detect the assertion of PCI_DEVSEL within four clock cycles following the address phase (five clock cycles after asserting PCI_FRAME), it terminates the transaction with a master-abort.

#### 16.4.2.8.2 Target-Initiated Termination

By asserting the PCI_STOP signal, a target may request that the initiator terminate the current transaction. Once asserted, the target holds PCI_STOP asserted until the initiator negates PCI_FRAME. Data may or may not be transferred during the request for termination. If PCI_TRDY and PCI_IRDY are asserted during the assertion of PCI_STOP, data is transferred. However, if PCI_TRDY is negated when PCI_STOP is asserted, it indicates that the target does not transfer any more data; therefore, the initiator does not wait for a final data transfer as it would in a completion termination.

When a transaction is terminated by PCI_STOP, the initiator must negate its REQ*n* signal for a minimum of two PCI clock cycles, (one corresponding to when the bus goes to the idle state (PCI_FRAME and PCI_IRDY negated)). If the initiator intends to complete the transaction, it can reassert its REQ*n* immediately following the two clock cycles. If the initiator does not intend to complete the transaction, it can assert REQ*n* whenever it needs to use the PCI bus again.

There are three types of target-initiated termination:

- Disconnect—Disconnect refers to termination requested because the target is temporarily unable to continue bursting. Disconnect implies that some data has been transferred. The initiator may restart the transaction at a later time starting with the address of the next untransferred data. (That is, data transfer may resume where it left off.)
- Retry—Retry refers to termination requested because the target is currently in a state where it is unable to process the transaction. Retry implies that no data was transferred. The initiator may start the entire transaction over again at a later time. Note that the *PCI Local Bus Specification*, Rev. 2.2 requires that all retried transactions must be completed.
- Target-Abort—Target-abort is an abnormal case of target-initiated termination. Target-abort is used when a fatal error has occurred or when a target can never respond.

As a target, the PCI controller terminates a transaction with a target disconnect due to the following:

- It is unable to respond within eight PCI clock cycles (not including the first data phase).
- The transaction is attempting to cross a 4-Kbyte boundary.
- A single beat of data has been transferred and the inbound ATMU is marked non-prefetchable.
- The end of a cache line has been transferred for a cache-wrap mode write transaction. See Section 16.4.2.3.1, “Memory Space Addressing,” for more information.

As a target, the PCI controller responds to a transaction with a retry due to the following:

- The 32-clock latency timer has expired, and the first data phase has not begun.
- There is no more internal buffer space available for an inbound transaction.

Target-abort is indicated by asserting $\overline{\text{PCI\_STOP}}$ and negating $\overline{\text{PCI\_DEVSEL}}$. This indicates that the target requires termination of the transaction and does not want the transaction retried. If a transaction is terminated by target-abort, the received target-abort bit (bit 12) of the initiator’s bus status register and the signaled target-abort bit (bit 11) of the target’s bus status register are set. Note that any data transferred in a target-aborted transaction may be corrupt.

For PCI writes to local memory, if an address parity error or data parity error occurs, the PCI controller aborts the transaction internally, but continues the transaction on the PCI bus.

Figure 16-53 shows several target-initiated terminations.

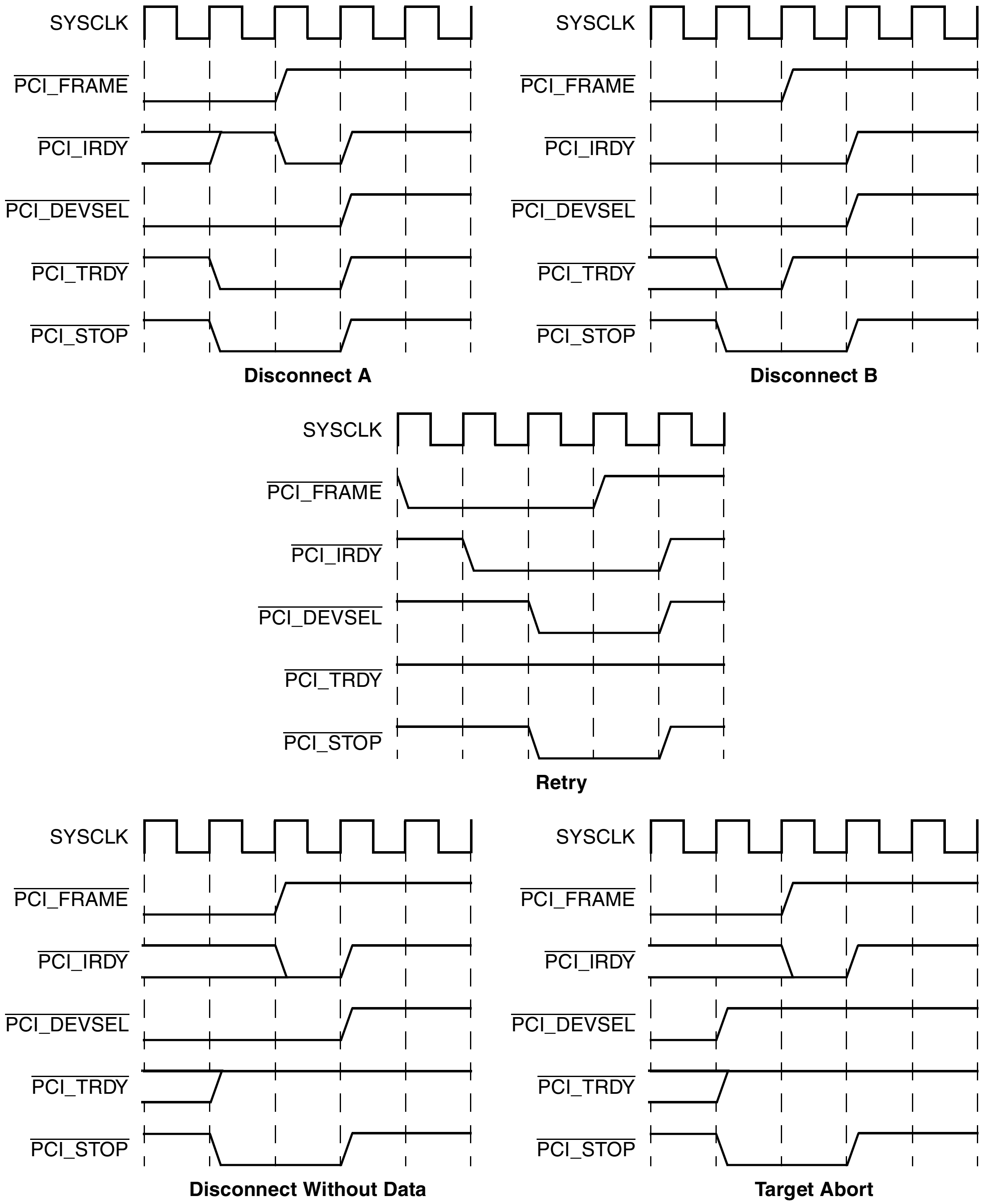


**Figure 16-53. PCI Target-Initiated Terminations**

The three disconnect terminations are unique in the data transferred at the end of the transaction. For disconnect A, the initiator is negating $\overline{\text{PCI\_IRDY}}$ when the target asserts $\overline{\text{PCI\_STOP}}$ and data is transferred only at the end of the current data phase. For disconnect B, the target negates $\overline{\text{PCI\_TRDY}}$ one clock after it asserts PCI_STOP, indicating that the target can accept the current data, but no more data can be transferred. For disconnect-without-data, the target asserts PCI_STOP when PCI_TRDY is negated indicating that the target cannot accept any more data.

### 16.4.2.9 Fast Back-to-Back Transactions

The PCI bus allows fast back-to-back transactions by the same master. During a fast back-to-back transaction, the initiator starts the next transaction immediately without an idle state. The last data phase completes when PCI_FRAME is negated, and PCI_IRDY and PCI_TRDY are asserted. The current master starts another transaction in the clock cycle immediately following the last data transfer for the previous transaction.

Fast back-to-back transactions must avoid contention on the PCI_TRDY, PCI_DEVSEL, PCI_PERR, and PCI_STOP signals. There are two types of fast back-to-back transactions—those that access the same target and those that access multiple targets sequentially. The first type places the burden of avoiding contention on the initiator; the second type places the burden of avoiding contention on all potential targets.

As an initiator, the PCI controller does not perform any fast back-to-back transactions. As a target, the PCI controller supports both types of fast back-to-back transactions.

During fast back-to-back transactions, the PCI controller monitors the bus states to determine if it is the target of a transaction. If the previous transaction was not directed to the PCI controller but the current transaction is directed at the PCI controller, it delays the assertion of PCI_DEVSEL (as well as PCI_TRDY, PCI_STOP, and PCI_PERR) for one clock cycle to allow the other target to stop driving the bus.

### 16.4.2.10 Dual Address Cycles

The PCI controller supports dual address cycle (DAC) commands (64-bit addressing on PCI bus) as both an initiator and a target. DACs are different from single address cycles (SACs) in that the address phase takes two PCI beats instead of one PCI beat to transfer (64-bit vs. 32-bit addressing). Only PCI memory commands can use DAC cycles; I/O, configuration, interrupt acknowledge, and special cycle command cannot use DAC cycles. The PCI controller block supports single-beat and burst DAC transactions.

For the case of the local processor, DAC generation depends on the setting of the POTEARx. If the POTEARx are programmed with nonzero values and a transaction from the local processor core hits in one of the outbound windows, a DAC transaction is generated on the PCI bus with the translated lower 32-bit addresses. Refer to Section 16.3.1.2, “PCI ATMU Outbound Registers,” for more information.

The timing sequence of the PCI signals for single-beat DAC reads is shown in Figure 16-54.

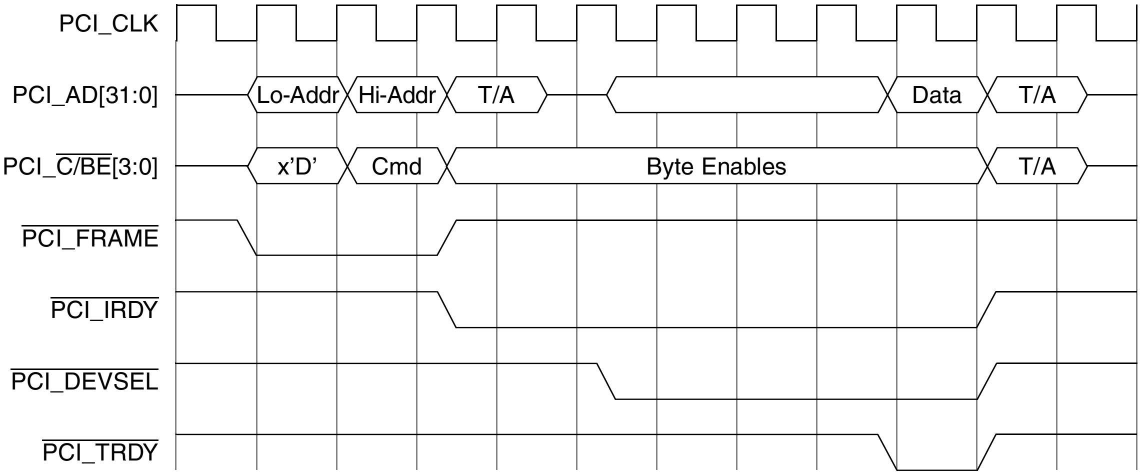


**Figure 16-54. DAC Single-Beat Read Example**

The timing for a DAC burst read is shown in Figure 16-55.

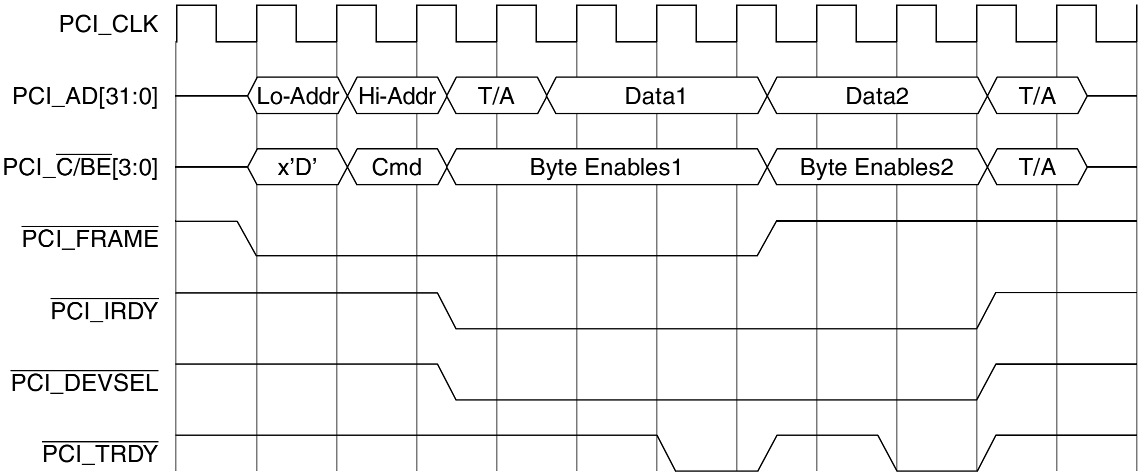


**Figure 16-55. DAC Burst Read Example**

Figure 16-56 and Figure 16-57 show timing examples for single-beat DAC writes and burst DAC writes, respectively.

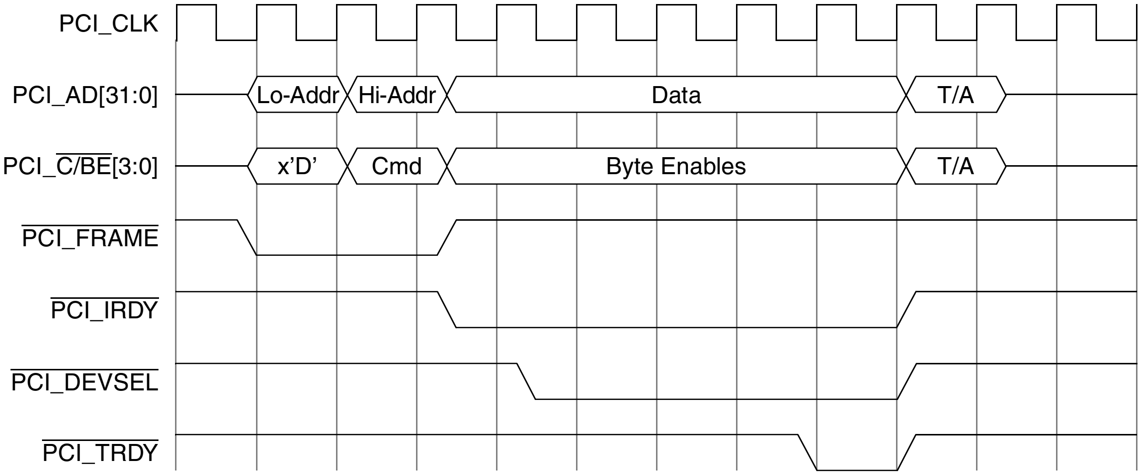


Figure 16-56. DAC Single-Beat Write Example

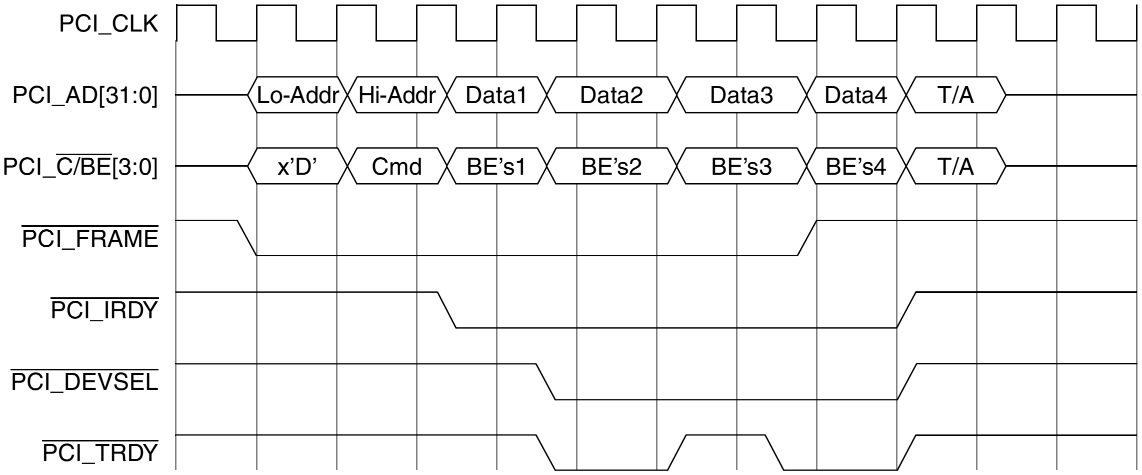


Figure 16-57. DAC Burst Write Example

### 16.4.2.11 Configuration Cycles

This section describes PCI configuration cycles used for configuring standard PCI devices. The PCI configuration space of any device is intended for configuration, initialization, and catastrophic error-handling functions only. Access to the PCI configuration space should be limited to initialization and error-handling software.

#### 16.4.2.11.1 PCI Configuration Space Header

The first 64 bytes of the 256-byte configuration space consists of a predefined header that every PCI device must support. The predefined header for all PCI devices is shown in Figure 16-58. The first 16 bytes of the predefined header are defined the same for all PCI devices; the remaining 48 bytes of the header may have differing layouts depending on the function of the device. Most PCI devices use the configuration header layout shown in Figure 16-58. The rest of the 256-byte configuration space is device-specific. The PCI header specific to the PCI controller is described in Section 16.3.2, “PCI Configuration Header.”

| | | | | Address Offset (Hex) |
|---|---|---|---|---|
| Device ID | | Vendor ID | | 00 |
| Status | | Command | | 04 |
| Class Code | | | Revision ID | 08 |
| BIST | Header Type | Latency Timer | Cache Line Size | 0C |
| Base Address Registers | | | | 10<br>14<br>18<br>1C<br>20<br>24 |
| Reserved | | | | 28 |
| Subsystem ID | | Subsystem Vendor ID | | 2C |
| Expansion ROM Base Address | | | | 30 |
| Reserved | | | | 34 |
| Reserved | | | | 38 |
| Max_Lat | Min_Gnt | Interrupt Pin | Interrupt Line | 3C |

**Figure 16-58. Standard PCI Configuration Header**

Table 16-49 summarizes the configuration header registers. Detailed descriptions of these registers are provided in the *PCI Local Bus Specification*, Rev. 2.2.

**Table 16-49. PCI Configuration Space Header Summary**

| Address Offset (Hex) | Register Name | Description |
|---|---|---|
| 0x00 | Vendor ID | Identifies the manufacturer of the device (assigned by the PCI SIG (special-interest group) to ensure uniqueness). |
| 0x02 | Device ID | Identifies the particular device (assigned by the vendor). |
| 0x04 | Command | Provides coarse control over a device’s ability to generate and respond to PCI bus cycles |
| 0x06 | Status | Records status information for PCI bus-related events |
| 0x08 | Revision ID | Specifies a device-specific revision code (assigned by vendor) |
| 0x09 | Class code | Identifies the generic function of the device and (in some cases) a specific register-level programming interface |
| 0x0C | Cache line size | Specifies the system cache line size in 32-bit units |
| 0x0D | Latency timer | Specifies the value of the latency timer in PCI bus clock units for the device when acting as an initiator |
| 0x0E | Header type | Bits 0–6 identify the layout of bytes 0x10–0x3F; bit 7 indicates a multifunction device. The most common header type (0x00) is shown in Figure 16-58 and in this table. |
| 0x0F | BIST | Optional register for control and status of built-in self test (BIST) |
| 0x10–0x27 | Base address registers | Address mapping information for memory and I/O space |

Table 16-49. PCI Configuration Space Header Summary (continued)

| Address Offset (Hex) | Register Name | Description |
|---|---|---|
| 0x28 | — | Reserved for future use |
| 0x2C | Subsystem Vendor ID | Identifies the subsystem vendor ID |
| 0x2E | Subsystem ID | Identifies the subsystem ID |
| 0x30 | Expansion ROM base address | Base address and size information for expansion ROM contained in an add-on board |
| 0x34, 0x38 | — | Reserved for future use |
| 0x3C | Interrupt line | Contains interrupt line routing information |
| 0x3D | Interrupt pin | Indicates which interrupt pin the device (or function) uses |
| 0x3E | Min_Gnt | Specifies the length of the device’s burst period in 0.25 μs units |
| 0x3F | Max_Lat | Specifies how often the device needs access to the bus in 0.25 μs units |

To access the configuration space, a 32-bit value must be written to the PCI CFG_ADDR register that specifies the target PCI bus, the target device on that bus, and the configuration register to be accessed within that device. A read or write to the PCI CFG_DATA register causes the host bridge to translate the access into a PCI configuration cycle (provided the enable bit in CONFIG_ADDR is set and the device number is not 0b1_1111).

See Section 16.3.1.1.1, “PCI Configuration Address Register (CFG_ADDR),” for details on PCI CFG_ADDR and Section 16.3.1.1.2, “PCI Configuration Data Register (CFG_DATA),” for details on PCI CFG_DATA.

### 16.4.2.11.2 Host Accessing the PCI Configuration Space

Power Architecture processor accesses to the PCI CFG_DATA register should use the load/store with byte-reversed instructions.

**Example:** Configuration sequence, 4-byte data read from the revision ID/standard programming interface/subclass code/class code registers at address offset 0x08 of the PCI configuration header (device 0 on the PCI bus 0 is the PCI controller itself).

```
Initial values:
         r0 contains 0x8000_0008
         r1 contains CCSRBAR + BlockBase + 0x000 (Address of PCI CFG_ADDR register)
         r2 contains CCSRBAR + BlockBase + 0x004 (Address of PCI CFG_DATA register)
         r3 contains 0xFFFF_FFFF
         Register at 0x08 contains 0x9988_7766 (0x0B to 0x08)
Code sequence:
         stw r0, 0 (r1)
         ld r3, 0 (r2)
Results:
         Address CCSRBAR + BlockBase + 0x000 contains 0x8000_0008
         Register r3 contains 0x6677_8899
```

### 16.4.2.11.3 Agent Accessing the PCI Configuration Space

When this device is configured as an agent device, it responds to a remote host-generated PCI configuration cycle. This is indicated by decoding the configuration command along with PCI's IDSEL being asserted. When the PCI controller detects an access to PCI CFG_DATA, it checks the enable flag and the device number in the PCI CFG_ADDR register. If the enable bit is set, and the device number is not 0b1_1111, the PCI controller performs a configuration cycle translation function and runs a configuration-read or configuration-write transaction on the PCI bus. The device number 0b1_1111 is used for performing interrupt-acknowledge and special-cycle transactions. See Section 16.4.2.12, "Other Bus Transactions," for more information. If the bus number corresponds to the local PCI bus (bus number = 0x00), the PCI controller performs a type 0 configuration cycle translation. If the bus number indicates a remote PCI bus (that is, nonlocal), the PCI controller performs a type 1 configuration cycle translation.

Note that in the following examples, the data in the configuration register is shown in little-endian order. This is because all the PCI registers are intrinsically little-endian. External PCI masters that use the local address map to access configuration space do not need to reverse bytes since byte lane redirection from the little-endian PCI bus is performed internally.

**Example:** Configuration sequence, 4-byte data write to PCI register at address offset 0x14 of Device 1 on PCI bus 0.

```
Initial values:
         r0 contains 0x8000_0814
         r1 contains CCSRBAR + BlockBase + 0x000 (Address of PCI CFG_ADDR register)
         r2 contains CCSRBAR + BlockBase + 0x004 (Address of PCI CFG_DATA register)
         r3 contains 0x1122_3344
         Register at 0x14 contains 0xFFFF_FFFF (0x17 to 0x14)
Code sequence:
         stw r0, 0 (r1) // Update PCI CFG_ADDR register to point to
                          //register offset 0x14 of device 1.
         stwbrx r3, 0 (r2)
Results:
         Address CCSRBAR + BlockBase + 0x000 contains 0x8000_0814
         Register at 0x14 contains 0x1122_3344 (0x17 to 0x14)
```

**Example:** Configuration sequence, 2-byte data write to PCI register at address offset 0x1C of Device 1 on PCI bus 0.

```
Initial values:
         r0 contains 0x8000_081C
         r1 contains CCSRBAR + BlockBase + 0x000
         r2 contains CCSRBAR + BlockBase + 0x004
         r3 contains 0xDDCC_BBAA
         Register at 0x1C contains 0xFFFF_FFFF (0x1F to 0x1C)
Code sequence:
         stw r0, 0 (r1)
         sthbrx r3, 0 (r2)
Results:
         Address CCSRBAR + BlockBase + 0x000 contains 0x8000_081C
         Register at 0x1C contains 0xFFFF_BBAA (0x1F to 0x1C)
```

#### 16.4.2.11.4 PCI Type 0 Configuration Translation

Figure 16-59 shows the PCI type 0 translation function performed on the contents of the PCI CFG_ADDR register to the PCI_AD[31:0] signals on the PCI bus during the address phase of the configuration cycle.

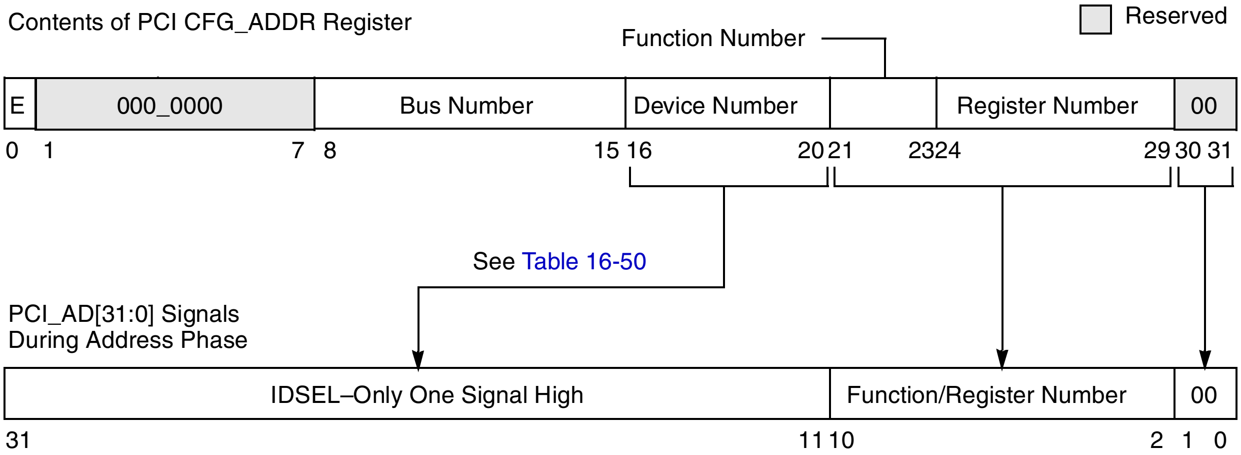


Figure 16-59. PCI Type 0 Configuration Translation

For PCI type 0 configuration cycles, the PCI controller translates the device number field of the PCI CFG_ADDR register into a unique IDSEL signal for up to 21 different devices. Each device connects its IDSEL input to one of the PCI_AD[31:11] signals. For PCI type 0 configuration cycles, the PCI controller translates the device number to AD*n* as shown in Table 16-50.

Table 16-50. PCI Type 0 Configuration—Device Number to AD*n* Translation

| Device Number | | AD*n* Used for IDSEL | Device Number | | AD*n* Used for IDSEL |
|---|---|---|---|---|---|
| Binary | Decimal | | Binary | Decimal | |
| 0b0_0000 | 0 | —[1] | 0b1_0100 | 20 | AD20 |
| 0b0_0001–0b0_1001 | 1–9 | —[2] | 0b1_0101 | 21 | AD21 |
| 0b0_1010 | 10 | AD31 | 0b1_0110 | 22 | AD22 |
| 0b0_1011 | 11 | AD11 | 0b1_0111 | 23 | AD23 |
| 0b0_1100 | 12 | AD12 | 0b1_1000 | 24 | AD24 |
| 0b0_1101 | 13 | AD13 | 0b1_1001 | 25 | AD25 |
| 0b0_1110 | 14 | AD14 | 0b1_1010 | 26 | AD26 |
| 0b0_1111 | 15 | AD15 | 0b1_1011 | 27 | AD27 |
| 0b1_0000 | 16 | AD16 | 0b1_1100 | 28 | AD28 |
| 0b1_0001 | 17 | AD17 | 0b1_1101 | 29 | AD29 |
| 0b1_0010 | 18 | AD18 | 0b1_1110 | 30 | AD30 |
| 0b1_0011 | 19 | AD19 | 0b1_1111 [3] | 31 | — |

[1] No external configuration transaction takes place; rather, internal registers are accessed.
[2] No IDSEL line asserted. Type0 configuration transaction is run, but ends with a master abort since no device responds.
[3] A device number of all ones indicates a PCI special-cycle or interrupt-acknowledge transaction.

For PCI type 0 translations, the function number and register number fields are copied without modification onto the PCI_AD[10:2] signals during the address phase. The PCI_AD[1:0] signals are driven to 0b00 during the address phase for type 0 configuration cycles. The PCI controller implements address stepping on configuration cycles so that the target's IDSEL, which is connected directly to one of the PCI_AD lines, reaches a stable value. This means that a valid address and command are driven on PCI_AD[31:0] and PCI_C/BE[3:0] one clock cycle before the assertion of PCI_FRAME.

#### 16.4.2.11.5 Type 1 Configuration Translation

For type 1 translations, the PCI controller copies the 30 high-order bits of the PCI CFG_ADDR register (without modification) onto the PCI_AD[31:2] signals during the address phase. The PCI controller automatically translates PCI_AD[1:0] into 0b01 during the address phase to indicate a type 1 configuration cycle.

### 16.4.2.12 Other Bus Transactions

There are two other PCI transactions that the PCI controller supports—interrupt acknowledge and special cycles. As an initiator, the PCI controller may initiate both interrupt acknowledge and special-cycle transactions; however, as a target, the PCI controller ignores interrupt-acknowledge and special-cycle transactions. Both transactions make use of the PCI CFG_ADDR and PCI CFG_DATA registers described in Section 16.4.2.11.3, "Agent Accessing the PCI Configuration Space."

#### 16.4.2.12.1 Interrupt-Acknowledge Transactions

The PCI bus supports an interrupt-acknowledge transaction. The interrupt-acknowledge command is a read operation implicitly addressed to the system interrupt controller. Note that the PCI interrupt-acknowledge command does not address the device's PIC processor interrupt-acknowledge register and does not return the interrupt vector address from the PIC unit. See Chapter 9, "Programmable Interrupt Controller (PIC)," for more information about the PIC unit.

When the PCI controller detects a read to the PCI CFG_DATA register, it checks the enable flag and the device number in the PCI CFG_ADDR register. If the enable bit is set, the bus number corresponds to the local PCI bus (bus number = 0x00), the device number is all ones (0b1_1111), the function number is all ones (0b111), and the register number is zero (0b00_0000), then the PCI controller performs an interrupt-acknowledge transaction. If the bus number indicates a nonlocal PCI bus, the PCI controller performs a type 1 configuration cycle translation, similar to any other configuration cycle for which the bus number does not match.

The address phase contains no valid information other than the interrupt-acknowledge command (PCI_C/BE[3:0] = 0b0000). Although there is no explicit address, PCI_AD[31:0] are driven to a stable state, and parity is generated. Only one device (the system interrupt controller) on the PCI bus should respond to the interrupt-acknowledge command by asserting PCI_DEVSEL. All other devices on the bus should ignore the interrupt-acknowledge command. As stated previously, the device's PIC unit does not respond to PCI interrupt-acknowledge commands.

During the data phase, the responding device returns the interrupt vector on PCI_AD[31:0] when PCI_TRDY is asserted. The size of the interrupt vector returned is indicated by the value driven on the PCI_C/BE[3:0] signals.

The PCI controller also provides a direct way to generate PCI interrupt-acknowledge transactions. Reads from PCI INT_ACK at offset 0x008 generate PCI interrupt-acknowledge transactions. Note that processor writes to these addresses do nothing.

#### 16.4.2.12.2 Special-Cycle Transactions

The special-cycle command provides a mechanism to broadcast select messages to all devices on the PCI bus. The special-cycle command contains no explicit destination address but is broadcast to all PCI agents.

When the PCI controller detects a write to PCI CFG_DATA, it checks the enable flag and the device number in PCI CFG_ADDR. If the enable bit is set, the bus number corresponds to the local PCI bus (bus number = 0x00), the device number is all ones (0b1_1111), the function number is all ones (0b111), and the register number is zero (0b00_0000), then the PCI controller performs a special-cycle transaction on the local PCI bus. If the bus number indicates a nonlocal PCI bus, the PCI controller performs a type 1 configuration cycle translation, similar to any other configuration cycle for which the bus number does not match.

Aside from the special-cycle command (PCI_C/$\overline{\text{BE}}$[3:0] = 0b0001) the address phase contains no other valid information. Although there is no explicit address, PCI_AD[31:0] are driven to a stable state, and parity is generated. During the data phase, PCI_AD[31:0] contain the special-cycle message and an optional data field. The special-cycle message is encoded on the 16 least-significant bits (PCI_AD[15:0]); the optional data field is encoded on the most-significant 16 lines (PCI_AD[31:16]). The special-cycle message encodings are assigned by the PCI SIG steering committee. The current list of defined encodings are provided in Table 16-51.

**Table 16-51. Special-Cycle Message Encodings**

| PCI_AD[15:0] | Message |
|---|---|
| 0x0000 | SHUTDOWN |
| 0x0001 | HALT |
| 0x0002 | x86 architecture-specific |
| 0x0003–0xFFFF | — |

Note that the PCI controller does not automatically issue a special-cycle message when it enters any of its power-saving modes. It is the responsibility of software to issue the appropriate special-cycle message, if needed.

Each receiving agent must determine whether the special-cycle message is applicable to itself. Assertion of $\overline{\text{PCI\_DEVSEL}}$ in response to a special-cycle command is not necessary. The initiator of the special-cycle transaction can insert wait states but since there is no specific target, the special-cycle message and optional data field are valid on the first clock $\overline{\text{PCI\_IRDY}}$ is asserted. All special-cycle transactions are terminated by master-abort; however, the master-abort bit in the initiator’s bus status register is not set for special-cycle terminations.

### 16.4.2.13 PCI Error Functions

PCI provides for parity and other system errors to be detected and reported. The PCI command register provides for selective enabling of specific PCI error detection. The PCI bus status register provides PCI error reporting. This section describes generation and detection of parity and error reporting for the PCI bus.

#### 16.4.2.13.1 PCI Parity

Generating parity is not optional; it must be performed by all PCI-compatible devices. All PCI transactions, regardless of type, calculate even parity; that is, the number of ones on the PCI_AD[31:0], PCI_C/$\overline{\text{BE}}$[3:0], and PCI_PAR signals all sum to an even number.

Parity provides a way to determine, on each transaction, if the initiator successfully addressed the target and transferred valid data. The PCI_C/$\overline{\text{BE}}$[3:0] signals are included in the parity calculation to ensure that the correct bus command is performed (during the address phase) and correct data is transferred (during the data phase). The agent responsible for driving the bus must also drive even parity on the PAR and PCI_PAR64 signal one clock cycle after a valid address phase or valid data transfer, as shown in Figure 16-60.

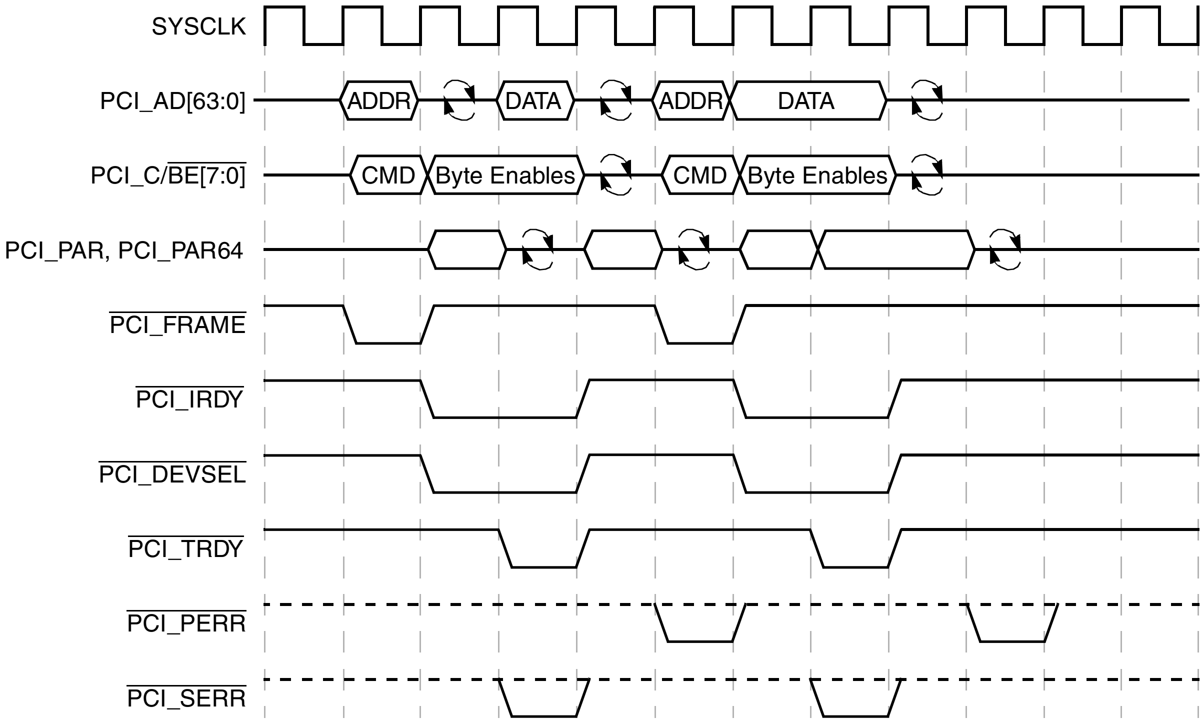


**Figure 16-60. PCI Parity Operation**

During the address and data phases, parity covers all 32 address/data signals and 4 command/byte enable signals, regardless of whether all lines carry meaningful information. Byte lanes not actually transferring data must contain stable (albeit meaningless) data and are included in parity calculation. During configuration, special-cycle, or interrupt-acknowledge commands; some address lines are not defined, but are driven to stable values and are included in parity calculation.

Agents that support parity checking must set the detected parity error bit in the PCI bus status register when a parity error is detected. Any additional response to a parity error is controlled by the parity error response bit in the PCI bus command register. If the parity error response bit is cleared, the agent ignores all parity errors.

#### 16.4.2.13.2 Error Reporting

PCI provides for the detection and signaling of both parity and other system errors. Two signals are used to report these errors—PCI_PERR and PCI_SERR. The PCI_PERR signal is used exclusively to report data parity errors on all transactions except special cycles. The PCI_SERR signal is used for other error signaling including address parity errors and data parity errors on special-cycle transactions; it may also be used to signal other system errors.

Table 16-52 shows the actions taken for each kind of error.

**Table 16-52. PCI Mode Error Actions**

| PCI Error Type | Error Detect Register bit | PCI Status Register bit | Comment |
|---|---|---|---|
| **PCI Outbound Read** | | | |
| Received SERR at any phase | Rcvd SERR | — | No data transferred |
| Received Parity Error for data phase | Mstr PERR | Detected Parity Error, Master Data Parity Error Detected | No data transferred |
| Master Abort | Mstr abort | Received Master Abort | No data transferred |
| Target Abort | Trgt abort | Received Target Abort | No data transferred |
| Memory space violation | ORMSV | — | No data transferred.<br>Only 8 bytes are requested in PCI bus |
| **PCI Outbound Write** | | | |
| Received SERR related to Address phase | Rcvd SERR | — | May float AD bus to avoid contention |
| Received SERR related to Data phase | Rcvd SERR | — | |
| Received PERR (Data phase) | Mstr PERR | Master Data Parity Error | |
| Master Abort | Mstr abort | Received Master Abort | |
| Target Abort | Trgt abort | Received Target Abort | |
| Memory space violation | OWMSV | — | Only 8 bytes transferred. |
| **PCI Inbound Read** | | | |
| Detected Parity Error for Address phase | Addr Parity Error | Detected Parity Error, Signaled System Error | Float AD bus |

Table 16-52. PCI Mode Error Actions (continued)

| PCI Error Type | Error Detect Register bit | PCI Status Register bit | Comment |
|---|---|---|---|
| Detected Parity Error on upper address bus for Address phase (SAC or DAC) | — | — | |
| Received $\overline{\text{SERR}}$ at any phase | Received $\overline{\text{SERR}}$ | — | |
| Received $\overline{\text{PERR}}$ (Data phase) | Target $\overline{\text{PERR}}$ | — | |
| Internal error | Target Abort | Signaled Target Abort | |
| **PCI Inbound Write** | | | |
| Detected Parity Error for Address phase | Addr Parity Error | Detected Parity Error,<br>Signaled System Error | Cache line purged |
| Detected Parity Error on upper address bus for Address phase (SAC or DAC) | — | — | |
| Received $\overline{\text{SERR}}$ at any phase | Rcvd $\overline{\text{SERR}}$ | — | |
| Detected Parity Error for Data phase | Trgt $\overline{\text{PERR}}$ | Detected Parity Error | Cache line purged |

## 16.5 Initialization/Application Information

This section describes some tips for use of the PCI controller.

### 16.5.1 Power-On Reset Configuration Modes

The PCI controller can power-on in three modes: host mode, agent mode and agent configuration lock mode. Certain bits in the configuration registers are set differently according to the POR (power-on reset) mode. Also, certain configuration bits have different implications when compared with past Freescale parts and PCI implementations. Note that after reset, the device cannot be switched from one mode to another.

The affected configuration bits are defined in Table 16-53.

Table 16-53. Affected Configuration Register Bits for POR

| Register (offset) | Bit | Name | Register Description |
|---|---|---|---|
| PCI Command Register (0x04) | 2 | Bus master | Controls whether the device can master a transaction on the PCI bus. If cleared, the device cannot master a transaction. This bit is independent of host or agent mode. |
| | 1 | Memory space | Controls the acknowledgement of inbound memory transactions. If cleared, all inbound memory accesses (including accesses to PCSRBAR space) end in a master abort. This bit is independent of host or agent mode. |

Table 16-53. Affected Configuration Register Bits for POR (continued)

| Register (offset) | Bit | Name | Register Description |
|---|---|---|---|
| PCI Bus Function Register (0x44) | 5 | ACL | Valid only in agent mode. Controls acknowledgement of inbound configuration accesses. If set, all inbound configuration accesses are retried. If cleared, inbound configuration accesses are acknowledged.<br>In host mode all inbound configuration accesses end in master aborts. |
| | 0 | PAH | Determines whether the device is in agent or host mode. Zero indicates host mode. |

The POR reset values for the affected configuration bits are described in Table 16-54.

Table 16-54. Power-On Reset Values for Affected Configuration Bits

| Mode | Configuration Bit | | | |
|---|---|---|---|---|
| | Bus Master | Memory Space | ACL | PAH |
| Host | 1 | 0 | X | 0 |
| Agent | 0 | 0 | 0 | 1 |
| Agent configuration lock | 0 | 0 | 1 | 1 |

### 16.5.1.1 Host Mode

When the device powers up in host mode, all inbound configuration accesses are ignored (and thus master aborted). The PBFR[ACL] bit is a don’t care. The device powers up with the ability to master transactions on the PCI bus, however in order to acknowledge memory transactions, the memory space bit must be set.

### 16.5.1.2 Agent Mode

When the device powers up in agent mode, it acknowledges inbound configuration accesses. However the device cannot master transactions or acknowledge inbound memory accesses on the PCI bus until the appropriate configuration bits (bus master and memory space, respectively) have been set.

### 16.5.1.3 Agent Configuration Lock Mode

Agent configuration lock mode is similar to agent mode with the added restriction that when the device powers up in agent configuration lock mode, it retries all inbound configuration accesses until the PBFR[ACL] bit is cleared. The purpose of this mode is to allow initial configuration on the port by the local processor before opening the port to be further configured by the external host. As in agent mode, the device in agent configuration lock mode cannot master transactions or acknowledge inbound memory accesses on the PCI bus until the appropriate configuration bits (bus master and memory space, respectively) have been set.

### 16.5.2 Byte Ordering

Whenever data must cross a bridge between two busses, the byte ordering of data on the source and destination buses must be considered. The internal platform bus of this device is inherently big endian and the PCI bus interface is inherently little endian.

There are two methods to handle ordering of data as it crosses a bridge—address invariance and data invariance. Address invariance preserves the addressing of bytes within a scalar data element, but not the relative significance of the bytes within that scalar. Conversely, data invariance preserves the relative significance of bytes within a scalar, but not the addressing of the individual bytes that make up a scalar.

This device uses address invariance as its byte ordering policy.

As stated above, address invariance preserves the byte address of each byte on an I/O interface as it is placed in memory or moved into a register. This policy can have the effect of reversing the significance order of bytes (most significant to least significant and vice versa), but it has the benefit of preserving the format of general data structures. Provided that software is aware of the endianness and format of the data structure, it can correctly interpret the data on either side of the bridge.

Figure 16-61 shows the transfer of a 4-byte scalar, 0x4142_4344, from a big endian source across an address invariant bridge to a little endian destination.

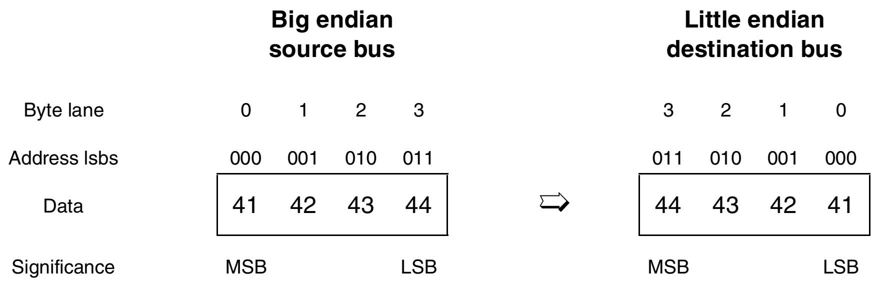


Figure 16-61. Address Invariant Byte Ordering—4 bytes Outbound

Note that although the significance of the bytes within the scalar have changed, the address of the individual bytes that make up the scalar have not changed. As long as software is aware that the source of the data used a big endian format, the data can be interpreted correctly.

Figure 16-63 shows data flowing the other way, from a little endian source to a big endian destination.

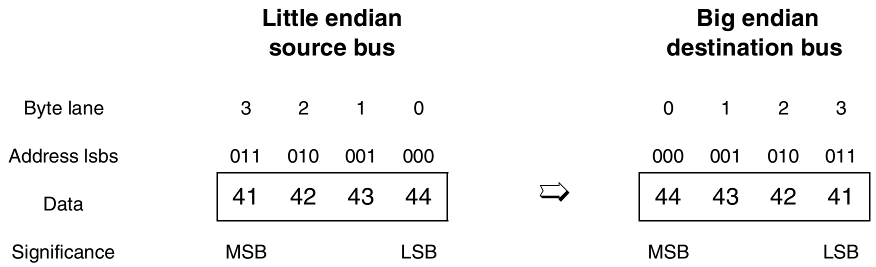


Figure 16-62. Address Invariant Byte Ordering—4 bytes Inbound

Figure 16-63 shows an outbound transfer of an 8-byte scalar, 0x5455_1617_CDCE_2728, using address invariance.

| | Big endian source bus | | | | | | | | | Little endian destination bus | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Byte lane | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Address lsbs | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 | | 111 | 110 | 101 | 100 | 011 | 010 | 001 | 000 |
| Data | 54 | 55 | 16 | 17 | CD | CE | 27 | 28 | ⇨ | 28 | 27 | CE | CD | 17 | 16 | 55 | 54 |
| Significance | MSB | | | | | | | LSB | | MSB | | | | | | | LSB |

**Figure 16-63. Address Invariant Byte Ordering—8 bytes Outbound**

Figure 16-64 shows an inbound transfer of a 2-byte scalar, 0x5837, using address invariance.

| | Little endian source bus | | | | | Big endian destination bus | | | |
|---|---|---|---|---|---|---|---|---|---|
| Byte lane | 3 | 2 | 1 | 0 | | 0 | 1 | 2 | 3 |
| Address lsbs | 011 | 010 | 001 | 000 | | 000 | 001 | 010 | 011 |
| Data | — | — | 58 | 37 | ⇨ | 37 | 58 | — | — |
| Significance | | | MSB | LSB | | MSB | LSB | | |

**Figure 16-64. Address Invariant Byte Ordering—2 bytes Inbound**

Note that in all of these examples, the original addresses of the individual bytes within the scalars (as created by the source) have been preserved.

### 16.5.2.1 Byte Order for Configuration Transactions

All internal memory-mapped registers in the CCSR space use big endian byte ordering. However, the PCI specification defines PCI configuration registers as little endian. All accesses to the PCI configuration port, CFG_DATA, including the those targeting the internal PCI configuration registers, use the address invariance policy as shown in Figure 16-65. Therefore, software must access CFG_DATA with little-endian formatted data—either using the **lwbrx**/**stwbrx** instructions or by manipulating the data before writing to and after reading from CFG_DATA.

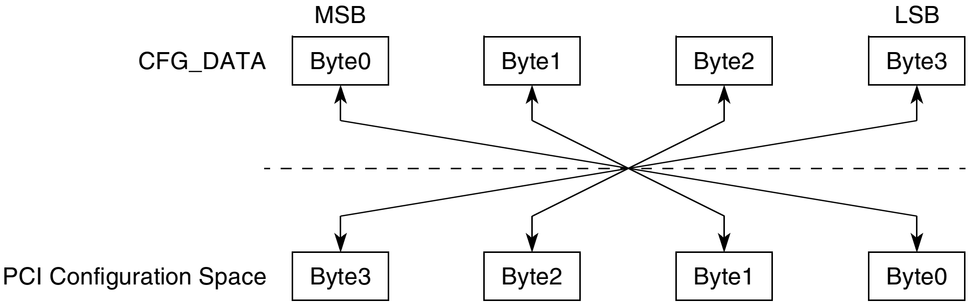


**Figure 16-65. CFG_DATA Byte Ordering**

# Chapter 17
# PCI Express Interface Controller

The PCI Express interface is compatible with the *PCI Express™ Base Specification, Revision 1.0a* (available from http://www.pcisig.org). It is beyond the scope of this manual to document the intricacies of the PCI Express protocol. This chapter describes the PCI Express controller of this device and provides a basic description of the PCI Express protocol. The specific emphasis is directed at how the device implements the PCI Express specification. Designers of systems incorporating PCI Express devices should refer to the specification for a thorough description of PCI Express.

**NOTE**

Much of the available PCI Express literature refers to a 16-bit quantity as a WORD and a 32-bit quantity as a DWORD. Note that this is inconsistent with the terminology in the rest of this manual where the terms ‘word’ and ‘double word’ refer to a 32-bit and 64-bit quantity, respectively. Where necessary to avoid confusion, the precise number of bits or bytes is specified.

## 17.1 Introduction

The PCI Express controller provides the mechanism to communicate with PCI Express devices. Figure 17-1 is a high-level block diagram of the PCI Express controller.

### 17.1.1 Overview

The PCI Express controller connects the internal platform to a 2.5- GHz serial interface. The MPC8536E offers one, two, or three PCI Express interfaces with up to x8 link width. (Note that the x8 link width is only available at CCB clock rates of 527 MHz or greater.)

As both an initiator and a target device, the PCI Express interface is capable of high-bandwidth data transfer and is designed to support next generation I/O devices. Upon coming out of reset, the PCI Express interface performs link width negotiation and exchanges flow control credits with its link partner. Once link autonegotiation is successful, the controller is in operation.

Internally, the design contains queues to keep track of inbound and outbound transactions. There is control logic that handles buffer management, bus protocol, transaction spawning and tag generation. In addition, there are memory blocks used to store inbound and outbound data.

The PCI Express controller can be configured to operate as either a PCI Express root complex (RC) or an endpoint (EP) device. An RC device connects the host CPU/memory subsystem to I/O devices while an EP device typically denotes a peripheral or I/O device. In RC mode, a PCI Express type 1 configuration header is used; in EP mode, a PCI Express type 0 configuration header is used.

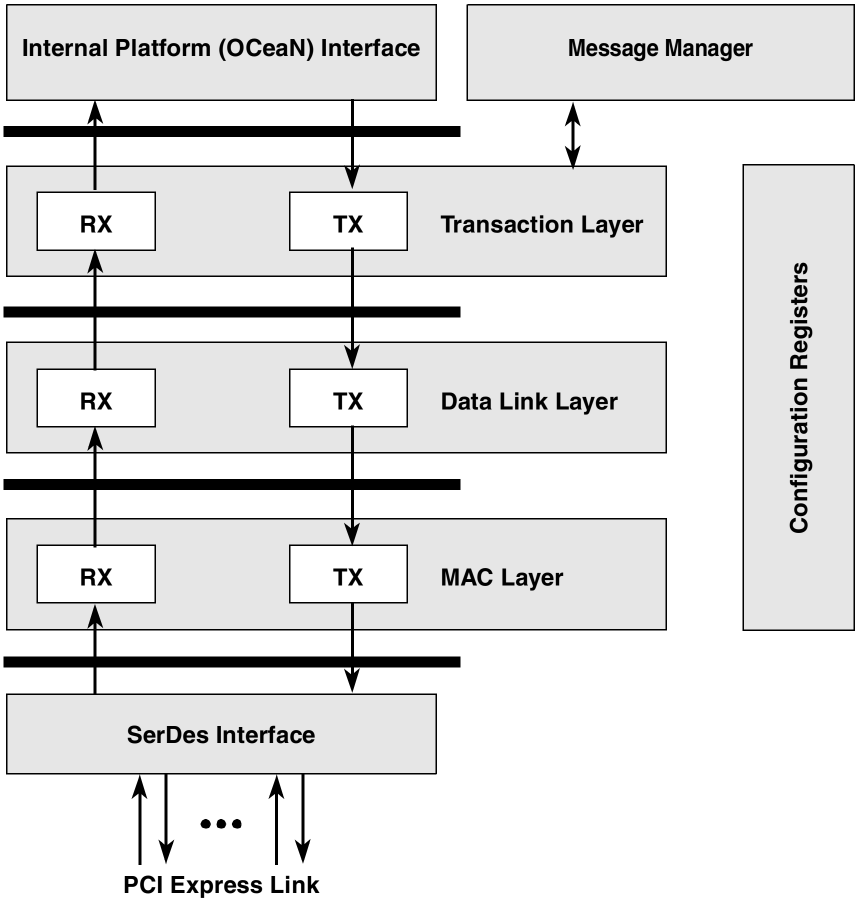


**Figure 17-1. PCI Express Controller Block Diagram**

As an initiator, the PCI Express controller supports memory read and write operations with a maximum transaction size of 256 bytes. In addition, configuration and I/O transactions are supported if the PCI Express controller is in RC mode. As a target interface, the PCI Express controller accepts read and write operations to local memory space. When configured as an EP device, the PCI Express controller accepts configuration transactions to the internal PCI Express configuration registers. Message generation and acceptance are supported in both RC and EP modes. Locked transactions and inbound I/O transactions are not supported.

### 17.1.1.1 Outbound Transactions

Outbound internal platform transactions to PCI Express are first mapped to a translation window to determine what PCI Express transactions are to be issued. A transaction from the internal platform can become a PCI Express Memory, I/O, Message, or Configuration transaction depending on the window attributes.

A transaction may be broken up into smaller sized transactions depending on the original request size, transaction type, and either the PCI Express device control register [MAX_PAYLOAD_SIZE] field for write requests or the PCI Express device control register [MAX_READ_SIZE] field for read requests. The controller performs PCI Express ordering rule checking to determine which transaction is to be sent on the PCI Express link.

In general, transactions are serviced in the order that they are received from the internal platform (OCeaN). Only when there is a stalled condition does the controller apply PCI Express ordering rules to outstanding transactions. For posted write transactions, once all data has been received from the internal platform (OCeaN), the data is forwarded to the PCI Express link and the transaction is considered as done. For non-posted write transactions, the controller waits for the completion packets to return before considering the transaction finished. For non-posted read transactions, the controller waits for all completion packets to return and then forwards all data back to the internal platform before terminating the transaction.

Note that after reset or when recovering from a link down condition, external transactions should not be attempted until the link has successfully trained. Software can poll the LTSSM state status register (PEX_LTSSM_STAT) to check the status of link training before issuing external requests.

### 17.1.1.2 Inbound Transactions

Inbound PCI Express transactions to internal platform are first mapped to a translation window to determine what internal platform transactions are to be issued.

A transaction may be broken up into smaller sized transactions when sending to the internal platform depending on the original request size, byte enables and starting/ending addresses. The controller performs PCI Express ordering rule checking to determine what transaction is to be sent next to the internal platform (OCeaN).

In general, transactions are serviced in the order that they are received from the PCI Express link. Only when there is a stalled condition does the controller apply PCI Express ordering to outstanding transactions. For posted write transactions, once all data has been received from the PCI Express link, the data is forwarded to the internal platform and the transaction is considered as done. For non-posted read transactions, the controller forwards internal platform data back to the PCI Express link.

Note that the controller splits transactions at the crossing of every 256-byte-aligned boundary when sending data back to the PCI Express link.

## 17.1.2 Features

The following is a list of features supported by the PCI Express controller:

- Compatible with the *PCI Express™ Base Specification, Revision 1.0a*
- Supports root complex (RC) and endpoint (EP) configurations
- 32- and 64-bit address support
- x8, x4, x2, and x1 link support. (x8 link width only available at CCB clock rates of 527 MHz or greater)
- Supports accesses to all PCI Express memory and I/O address spaces (requestor only)
- Supports posting of processor-to-PCI Express and PCI Express-to-memory writes
- Supports strong and relaxed transaction ordering rules
- PCI Express configuration registers (type 0 in EP mode, type 1 in RC mode)

- Baseline and advanced error reporting support
- One virtual channel (VC0)
- 256-byte maximum payload size (MAX_PAYLOAD_SIZE)
- Supports three inbound general-purpose translation windows and one configuration window
- Supports four outbound translation windows and one default window
- Supports eight non-posted and four posted PCI Express transactions
- Supports up to six priority 0 internal platform reads and eight priority 0 to 2 internal platform writes. (The maximum number of outstanding transactions at any given time is eight.)
- Credit-based flow control management
- Supports PCI Express messages and interrupts
- Accepts up to 256-byte transactions from the internal platform (OCeaN)

### 17.1.3 Modes of Operation

Several parameters that affect the PCI Express controller modes of operation are determined at power-on reset (POR) by reset configuration signals as described in Chapter 4, "Reset, Clocking, and Initialization."

**Table 17-1. POR Parameters for PCI Express Controller**

| Parameter | Description | Section/Page |
|---|---|---|
| Host/Agent Configuration | Selects between root complex (RC) and endpoint (EP) modes. | 4.4.3.7/4-15 |
| I/O Port Selection | Selects the width of the PCI Express links | 4.4.3.8/4-16 |

#### 17.1.3.1 Root Complex/Endpoint Modes

The PCI Express controller can function as either a root complex (RC) or an endpoint (EP) on the PCI Express link. The host/agent configuration input signals cfg_host_agt[0:2] determine the RC/EP mode.

#### 17.1.3.2 Link Width

The MPC8536E initial link widths are determined by POR configuration signals as described in Section 17.1.3, "Modes of Operation." Note that the x8 link width is only available at CCB clock rates of 527 MHz or greater.

## 17.2 External Signal Descriptions

PCI Express defines the connection between two devices as a link, which can be composed of a single or multiple lanes. Each lane consists of a differential pair for transmitting (TX*n* and $\overline{\text{TX}n}$) and a differential pair for receiving (RX*n* and $\overline{\text{RX}n}$) with an embedded data clock.

Although the generic PCI Express controller described here accommodates up to a single ×8 link, there are three PCI Express controllers instantiated on the MPC8536E. Please refer to Section 4.4.3.8, "SerDes1 I/O

Port Selection," for specific pin muxing details. Note that the x8 link width is only available at CCB clock rates of 527 MHz or greater.

Table 17-2 contains detailed descriptions of the external PCI Express interface signals.

**Table 17-2. PCI Express Interface Signals—Detailed Signal Descriptions**

| Signal | I/O | Description | |
|---|---|---|---|
| SD1_RX[7:0] | I | Receive data. The receive data signals carry PCI Express packet information. PCI Express signals may appear as follows:<br>• PCI Express 1: SD1_RX[7:0] or SD1_RX[3:0]<br>• PCI Express 2: SD1_RX[7:4] or SD1_RX[5:4]<br>PCI Express 3: SD1_RX[7:6] | |
| | | **State Meaning** | Asserted/Negated—Represents data being received from the PCI Express interface. |
| | | **Timing** | Assertion/Negation—As described in the *PCI Express Base Specification, Revision 1.0a*. |
| $\overline{\text{SD1\_RX}}$[7:0] | I | Receive data, inverted. $\overline{\text{SD1\_RX}}$[7:0]are the inverted forms of the receive data signals (SD1_RX[7:0]). | |
| | | **State Meaning** | Asserted/Negated—Represents the inverse of data being received from the PCI Express interface. |
| | | **Timing** | Assertion/Negation—As described in the *PCI Express Base Specification, Revision 1.0a*. |
| SD1_TX[7:0] | O | Transmit data. The transmit data signals carry PCI Express packet information. PCI Express signals may appear as follows:<br>• PCI Express 1: SD1_RX[7:0] or SD1_RX[3:0]<br>• PCI Express 2: SD1_RX[7:4] or SD1_RX[5:4]<br>PCI Express 3: SD1_RX[7:6] | |
| | | **State Meaning** | Asserted/Negated—Represents data being transmitted to the PCI Express interface. |
| | | **Timing** | Assertion/Negation—As described in the *PCI Express Base Specification, Revision 1.0a* . |
| $\overline{\text{SD1\_TX}}$[7:0] | O | Transmit data, inverted. $\overline{\text{SD1\_TX}}$[7:0] are the inverted form of the transmit data signals (SD1_TX[7:0]). | |
| | | **State Meaning** | Asserted/Negated—Represents the inverse of data being transmitted to the PCI Express interface. |
| | | **Timing** | Assertion/Negation—As described in the *PCI Express Base Specification, Revision 1.0a*. |

## 17.3 Memory Map/Register Definitions

The PCI Express interface supports the following register types:

- Memory-mapped registers—these registers control PCI Express address translation, PCI error management, and PCI Express configuration register access. These registers are described in Section 17.3.1, "PCI Express Memory Mapped Registers," and its subsections.
- PCI Express configuration registers contained within the PCI Express configuration space—these registers are specified by the PCI Express specification for every PCI Express device. These registers are described in Section 17.3.7, "PCI Express Configuration Space Access," and its subsections.

### 17.3.1 PCI Express Memory Mapped Registers

The PCI Express memory mapped registers are accessed by reading and writing to an address comprised of the base address (specified in the CCSRBAR on the local side or the PEXCSRBAR on the PCI Express side) plus the block base address, plus the offset of the specific register to be accessed. Note that all memory-mapped registers (except the PCI Express configuration data register, PEX_CONFIG_DATA) must only be accessed as 32-bit quantities.

Also note that although the table explicitly lists only the registers for the PCI Express controller 1, the register map for PCI Express controllers 2 and 3 are the same except for the block base address. Memory-mapped registers for PCI Express controller 1 begin at block base address 0x0_A000, controller 2 registers begin at 0x0_9000, and controller 3 registers begin at 0x0_B000.

Table 17-3 lists the memory-mapped registers. In this table and in the register figures and field descriptions, the following access definitions apply:

- Reserved fields are always ignored for the purposes of determining access type.
- R/W, R, and W (read/write, read only, and write only) indicate that all the non-reserved fields in a register have the same access type.
- w1c indicates that all of the non-reserved fields in a register are cleared by writing ones to them.
- Mixed indicates a combination of access types.
- Special is used when no other category applies. In this case the register figure and field description table should be read carefully.

**Table 17-3. PCI Express Memory-Mapped Register Map**

| PCI Express Controller 1 —Block Base Address 0x0_A000<br>PCI Express Controller 2—Block Base Address 0x0_9000<br>PCI Express Controller 3—Block Base Address 0x0_B000 | | | | |
|---|---|---|---|---|
| **Offset** | **Register** | **Access** | **Reset** | **Section/Page** |
| **PCI Express Controller 1 Memory-Mapped Registers** | | | | |
| **PCI Express Configuration Access Registers** | | | | |
| 0x000 | PEX_CONFIG_ADDR—PCI Express configuration address register | R/W | 0x0000_0000 | 17.3.2.1/17-10 |
| 0x004 | PEX_CONFIG_DATA—PCI Express configuration data register | R/W | 0x0000_0000 | 17.3.2.2/17-10 |
| 0x008 | Reserved | — | — | |
| 0x00C | PEX_OTB_CPL_TOR—PCI Express outbound completion timeout register | R/W | 0x0010_FFFF | 17.3.2.3/17-11 |
| 0x010 | PEX_CONF_RTY_TOR—PCI Express configuration retry timeout register | R/W | 0x0400_FFFF | 17.3.2.4/17-12 |
| 0x014 | PEX_CONFIG—PCI Express configuration register | R/W | 0x0000_0000 | 17.3.2.5/17-12 |
| 0x018–0x01C | Reserved | — | — | |
| **PCI Express Power Management Event & Message Registers** | | | | |
| 0x020 | PEX_PME_MES_DR—PCI Express PME & message detect register | w1c | 0x0000_0000 | 17.3.3.1/17-13 |
| 0x024 | PEX_PME_MES_DISR—PCI Express PME & message disable register | R/W | 0x0000_0000 | 17.3.3.2/17-15 |

Table 17-3. PCI Express Memory-Mapped Register Map (continued)

| PCI Express Controller 1 —Block Base Address 0x0_A000<br>PCI Express Controller 2—Block Base Address 0x0_9000<br>PCI Express Controller 3—Block Base Address 0x0_B000 | | | | |
|---|---|---|---|---|
| **Offset** | **Register** | **Access** | **Reset** | **Section/Page** |
| 0x028 | PEX_PME_MES_IER—PCI Express PME & message interrupt enable register | R/W | 0x0000_0000 | 17.3.3.3/17-16 |
| 0x02C | PEX_PMCR—PCI Express power management command register | R/W | 0x0000_0000 | 17.3.3.4/17-18 |
| 0x030–0xBF4 | Reserved | — | — | |
| **PCI Express IP Block Revision Registers** | | | | |
| 0xBF8 | IP block revision register 1 (PEX_IP_BLK_REV1) | R | 0x0208_0100 | 17.3.4.1/17-19 |
| 0xBFC | IP block revision register 2 (PEX_IP_BLK_REV2) | R | 0x0000_0000 | 17.3.4.2/17-19 |
| **PCI Express ATMU Registers** | | | | |
| **Outbound Window 0 (Default)** | | | | |
| 0xC00 | PEXOTAR0—PCI Express outbound translation address register 0 (default) | R/W | 0x0000_0000 | 17.3.5.1.1/17-20 |
| 0xC04 | PEXOTEAR0—PCI Express outbound translation extended address register 0 (default) | R/W | 0x0000_0000 | 17.3.5.1.2/17-21 |
| 0xC08–0xC0C | Reserved | — | — | |
| 0xC10 | PEXOWAR0—PCI Express outbound window attributes register 0 (default) | Mixed | 0x8004_4023 | 17.3.5.1.4/17-22 |
| 0xC14–0xC1C | Reserved | — | — | |
| **Outbound Window 1** | | | | |
| 0xC20 | PEXOTAR1—PCI Express outbound translation address register 1 | R/W | 0x0000_0000 | 17.3.5.1.1/17-20 |
| 0xC24 | PEXOTEAR1—PCI Express outbound translation extended address register 1 | R/W | 0x0000_0000 | 17.3.5.1.2/17-21 |
| 0xC28 | PEXOWBAR1—PCI Express outbound window base address register 1 | R/W | 0x0000_0000 | 17.3.5.1.3/17-22 |
| 0xC2C | Reserved | — | — | |
| 0xC30 | PEXOWAR1—PCI Express outbound window attributes register 1 | R/W | 0x0004_4023 | 17.3.5.1.4/17-22 |
| 0xC34–0xC3C | Reserved | — | — | |
| **Outbound Window 2** | | | | |
| 0xC40 | PEXOTAR2—PCI Express outbound translation address register 2 | R/W | 0x0000_0000 | 17.3.5.1.1/17-20 |
| 0xC44 | PEXOTEAR2—PCI Express outbound translation extended address register 2 | R/W | 0x0000_0000 | 17.3.5.1.2/17-21 |
| 0xC48 | PEXOWBAR2—PCI Express outbound window base address register 2 | R/W | 0x0000_0000 | 17.3.5.1.3/17-22 |
| 0xC4C | Reserved | — | — | |
| 0xC50 | PEXOWAR2—PCI Express outbound window attributes register 2 | R/W | 0x0004_4023 | 17.3.5.1.4/17-22 |

**Table 17-3. PCI Express Memory-Mapped Register Map (continued)**

| PCI Express Controller 1 —Block Base Address 0x0_A000<br>PCI Express Controller 2—Block Base Address 0x0_9000<br>PCI Express Controller 3—Block Base Address 0x0_B000 | | | | |
|---|---|---|---|---|
| **Offset** | **Register** | **Access** | **Reset** | **Section/Page** |
| 0xC54–<br>0xC5C | Reserved | — | — | |
| **Outbound Window 3** | | | | |
| 0xC60 | PEXOTAR3—PCI Express outbound translation address register 3 | R/W | 0x0000_0000 | 17.3.5.1.1/17-20 |
| 0xC64 | PEXOTEAR3—PCI Express outbound translation extended address register 3 | R/W | 0x0000_0000 | 17.3.5.1.2/17-21 |
| 0xC68 | PEXOWBAR3—PCI Express outbound window base address register 3 | R/W | 0x0000_0000 | 17.3.5.1.3/17-22 |
| 0xC6C | Reserved | — | — | |
| 0xC70 | PEXOWAR3—PCI Express outbound window attributes register 3 | R/W | 0x0004_4023 | 17.3.5.1.4/17-22 |
| 0xC74–<br>0xC7C | Reserved | — | — | |
| **Outbound Window 4** | | | | |
| 0xC80 | PEXOTAR4—PCI Express outbound translation address register 4 | R/W | 0x0000_0000 | 17.3.5.1.1/17-20 |
| 0xC84 | PEXOTEAR4—PCI Express outbound translation extended address register 4 | R/W | 0x0000_0000 | 17.3.5.1.2/17-21 |
| 0xC88 | PEXOWBAR4—PCI Express outbound window base address register 4 | R/W | 0x0000_0000 | 17.3.5.1.3/17-22 |
| 0xC8C | Reserved | — | — | |
| 0xC90 | PEXOWAR4—PCI Express outbound window attributes register 4 | R/W | 0x0004_4023 | 17.3.5.1.4/17-22 |
| 0xC94–<br>0xC9C | Reserved | — | — | |
| 0xD14–<br>0xD9C | Reserved | — | — | |
| **Inbound Window 3** | | | | |
| 0xDA0 | PEXITAR3—PCI Express inbound translation address register 3 | R/W | 0x0000_0000 | 17.3.5.2.3/17-26 |
| 0xDA4 | Reserved | — | — | |
| 0xDA8 | PEXIWBAR3—PCI Express inbound window base address register 3 | R/W | 0x0000_0000 | 17.3.5.2.4/17-27 |
| 0xDAC | PEXIWBEAR3—PCI Express inbound window base extended address register 3 | R/W | 0x0000_0000 | 17.3.5.2.5/17-27 |
| 0xDB0 | PEXIWAR3—PCI Express inbound window attributes register 3 | R/W | 0x20F4_4023 | 17.3.5.2.6/17-28 |
| 0xDB4–<br>0xDBC | Reserved | — | — | |
| **Inbound Window 2** | | | | |
| 0xDC0 | PEXITAR2—PCI Express inbound translation address register 2 | R/W | 0x0000_0000 | 17.3.5.2.3/17-26 |
| 0xDC4 | Reserved | — | — | |
| 0xDC8 | PEXIWBAR2—PCI Express inbound window base address register 2 | R/W | 0x0000_0000 | 17.3.5.2.4/17-27 |

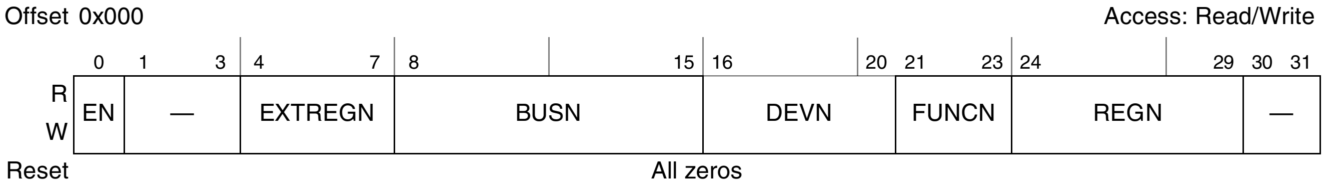
### Table 17-3. PCI Express Memory-Mapped Register Map (continued)

| PCI Express Controller 1 —Block Base Address 0x0_A000<br>PCI Express Controller 2—Block Base Address 0x0_9000<br>PCI Express Controller 3—Block Base Address 0x0_B000 | | | | |
|---|---|---|---|---|
| **Offset** | **Register** | **Access** | **Reset** | **Section/Page** |
| 0xDCC | PEXIWBEAR2—PCI Express inbound window base extended address register 2 | R/W | 0x0000_0000 | 17.3.5.2.5/17-27 |
| 0xDD0 | PEXIWAR2—PCI Express inbound window attributes register 2 | R/W | 0x20F4_4023 | 17.3.5.2.6/17-28 |
| 0xDD4–<br>0xDDC | Reserved | — | — | |
| **Inbound Window 1** | | | | |
| 0xDE0 | PEXITAR1—PCI Express inbound translation address register 1 | R/W | 0x0000_0000 | 17.3.5.2.3/17-26 |
| 0xDE4 | Reserved | — | — | |
| 0xDE8 | PEXIWBAR1—PCI Express inbound window base address register 1 | R/W | 0x0000_0000 | 17.3.5.2.4/17-27 |
| 0xDEC | Reserved | — | — | |
| 0xDF0 | PEXIWAR1—PCI Express inbound window attributes register 1 | R/W | 0x20F4_4023 | 17.3.5.2.6/17-28 |
| 0xDF4–<br>0xDFC | Reserved | — | — | |
| **PCI Express Error Management Registers** | | | | |
| 0xE00 | PEX_ERR_DR—PCI Express error detect register | w1c | 0x0000_0000 | 17.3.6.1/17-30 |
| 0xE04 | Reserved | — | — | — |
| 0xE08 | PEX_ERR_EN—PCI Express error interrupt enable register | R/W | 0x0000_0000 | 17.3.6.2/17-32 |
| 0xE0C | Reserved | — | — | — |
| 0xE10 | PEX_ERR_DISR—PCI Express error disable register | R/W | 0x0000_0000 | 17.3.6.3/17-34 |
| 0xE14–<br>0xE1C | Reserved | — | — | — |
| 0xE20 | PEX_ERR_CAP_STAT—PCI Express error capture status register | Mixed | 0x0000_0000 | 17.3.6.4/17-36 |
| 0xE24 | Reserved | — | — | — |
| 0xE28 | PEX_ERR_CAP_R0—PCI Express error capture register 0 | R/W | 0x0000_0000 | 17.3.6.5/17-36 |
| 0xE2C | PEX_ERR_CAP_R1—PCI Express error capture register 1 | R/W | 0x0000_0000 | 17.3.6.6/17-38 |
| 0xE30 | PEX_ERR_CAP_R2—PCI Express error capture register 2 | R/W | 0x0000_0000 | 17.3.6.7/17-40 |
| 0xE34 | PEX_ERR_CAP_R3—PCI Express error capture register 3 | R/W | 0x0000_0000 | 17.3.6.8/17-42 |
| 0xE38–<br>0xFFC | Reserved | — | — | |
| **PCI Express Controller 2 Memory-Mapped Registers** | | | | |
| 0x000–<br>0xFFC | PCI Express Controller 2 registers<br>**Note:** All registers defined for PCI Express Controller 1 are also defined for PCI Express Controller 2; the offsets of PCI Express Controller 2 registers are the same except they have a different block base address. | | | |
| **PCI Express Controller 3 Memory-Mapped Registers** | | | | |
| 0x000–<br>0xFFC | PCI Express Controller 3 registers<br>**Note:** All registers defined for PCI Express Controller 1 are also defined for PCI Express Controller 3; the offsets of PCI Express Controller 3 registers are the same except they have a different block base address. | | | |

## 17.3.2 PCI Express Configuration Access Registers

### 17.3.2.1 PCI Express Configuration Address Register (PEX_CONFIG_ADDR)

The PCI Express configuration address register, shown in Figure 17-2, contains address information for accesses to PCI Express internal and external configuration registers.



Offset 0x000 Access: Read/Write

| | 0 | 1 3 | 4 7 | 8 15 | 16 20 | 21 23 | 24 29 | 30 31 |
|---|---|---|---|---|---|---|---|---|
| R<br>W | EN | — | EXTREGN | BUSN | DEVN | FUNCN | REGN | — |
| Reset | All zeros | | | | | | | |

**Figure 17-2. PCI Express Configuration Address Register (PEX_CONFIG_ADDR)**

The fields of the PCI Express configuration address register are described in Table 17-4.

**Table 17-4. PEX_CONFIG_ADDR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | EN | Enable. This bit allows a PCI Express configuration access when PEX_CONFIG_DATA is accessed. If this bit is cleared, writing to PEX_CONFIG_DATA has no effect and reading PEX_CONFIG_DATA returns unknown data. |
| 1–3 | — | Reserved |
| 4–7 | EXTREGN | Extended register number. This field allows access to extended PCI Express configuration space (that is, the registers in the offset range from 0x100 to 0xFFF). |
| 8–15 | BUSN | Bus number. PCI bus number to access |
| 16–20 | DEVN | Device number. Device number to access on specified bus |
| 21–23 | FUNCN | Function number. Function to access within specified device |
| 24–29 | REGN | Register number. 32-bit register to access within specified device |
| 30–31 | — | Reserved |

Both root complex (RC) and endpoint (EP) configuration headers contain 4096 bytes of address space. To access a register within the header, both the extended register number and the register number fields are concatenated to form the 4-byte aligned address of the register. That is, the register address is extended register number || register number || 0b00.

### 17.3.2.2 PCI Express Configuration Data Register (PEX_CONFIG_DATA)

The PCI Express configuration data register, show in Figure 17-3, is a 32-bit port for internal and external configuration access. Note that accesses of 1, 2, or 4 bytes to the PCI Express configuration data register are allowed. Also note that accesses to the little-endian PCI Express configuration space must be properly formatted. See Section 17.4.1.2.1, “Byte Order for Configuration Transactions,” for more information.

Offset 0x004 Access: Read/Write

| | 0–31 |
|---|---|
| R<br>W | Data |
| Reset | All zeros |

**Figure 17-3. PCI Express Configuration Data Register (PEX_CONFIG_DATA)**

The fields of the PCI Express configuration data register are described in Table 17-5.

**Table 17-5. PEX_CONFIG_DATA Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–31 | Data | A read or write to this register starts a PCI Express configuration cycle if the PEX_CONFIG_ADDR enable bit is set (PEX_CONFIG_ADDR[EN] = 1). |

## 17.3.2.3 PCI Express Outbound Completion Timeout Register (PEX_OTB_CPL_TOR)

The PCI Express outbound completion timeout register, shown in Figure 17-4, contains the maximum wait time for a response to come back as a result of an outbound non-posted request before a timeout condition occurs.

Offset 0x00C Access: Read/Write

| | 0 | 1–7 | 8–31 |
|---|---|---|---|
| R<br>W | TD | — | TC |
| Reset | 0 | 0 0 0 0 0 0 0 | 0 0 0 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 |

**Figure 17-4. PCI Express Outbound Completion Timeout Register (PEX_OTB_CPL_TOR)**

The fields of the PCI Express outbound completion timeout register are described in Table 17-6.

**Table 17-6. PEX_OTB_CPL_TOR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | TD | Timeout disable. This bit controls the enabling/disabling of the timeout function.<br>0 Enable completion timeout<br>1 Disable completion timeout |
| 1–7 | — | Reserved |
| 8–31 | TC | Timeout counter. This is the value that is used to load the response counter of the completion timeout. One TC unit is 8× the PCI Express controller clock period; that is, one TC unit is 20 ns at 400 MHz, and 30 ns at 266.66 MHz.<br>The following are examples of timeout periods based on different TC settings:<br>0x00_0000 Reserved<br>0x10_FFFF 22.28 ms at 400 MHz controller clock; 33.34 ms at 266.66 MHz controller clock<br>0xFF_FFFF335.54 ms at 400 MHz controller clock; 503.31 ms at 266.66 MHz controller clock |

### 17.3.2.4 PCI Express Configuration Retry Timeout Register (PEX_CONF_RTY_TOR)

The PCI Express configuration retry timeout register, shown in Figure 17-5, contains the maximum time period during which retries of configuration transactions which resulted in a CRS response occur.

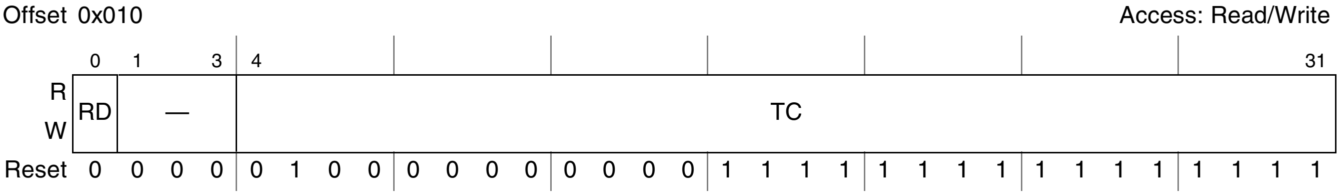


**Figure 17-5. PCI Express Configuration Retry Timeout Register (PEX_CONF_RTY_TOR)**

The fields of the PCI Express configuration retry timeout register are described in Table 17-7.

**Table 17-7. PEX_CONF_RTY_TOR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | RD | Retry disable. This bit disables the retry of a configuration transaction that receives a CRS status response packet.<br>0 Enable retry of a configuration transaction in response to receiving a CRS status response until the timeout counter (defined by the PEX_CONF_RTY_TOR[TC] field) has expired.<br>1 Disable retry of a configuration transaction regardless of receiving a CRS status response. |
| 1–3 | — | Reserved |
| 4–31 | TC | Timeout counter. This is the value that is used to load the CRS response counter.<br>One TC unit is 8x the PCI Express controller clock period; that is, one TC unit is 20 ns at 400 MHz and 30 ns at 266.66 MHz.<br>Timeout period based on different TC settings:<br>0x000_0000 Reserved<br>0x400_FFFF 1.34 s at 400 MHz controller clock, 2.02 s at 266.66 MHz controller clock<br>0xFFF_FFFF 5.37 s at 400 MHz controller clock, 8.05 s at 266.66 MHz controller clock |

### 17.3.2.5 PCI Express Configuration Register (PEX_CONFIG)

The PCI Express configuration register, shown in Figure 17-6, contains various control switches for the controller.

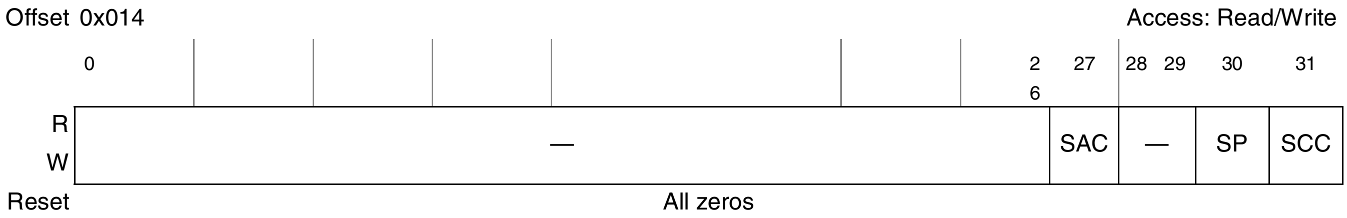


**Figure 17-6. PCI Express Configuration Register (PEX_CONFIG)**

The fields of the PCI Express configuration register are described in Table 17-8.

Table 17-8. PEX_CONFIG Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–26 | — | Reserved |
| 27 | SAC | Sense ASPM Control. This bit controls the default value of ASPM of PEX Link Control Register's bit 0. See Section 17.3.9.11, "PCI Express Link Control Register—0x5C," for more information. |
| 28–29 | — | Reserved |
| 30 | SP | Slot Present. This bit controls the default value of the PCI Express capabilities register [slot] bit. See Section 17.3.9.6, "PCI Express Capabilities Register—0x4E," for more information. |
| 31 | SCC | Slot Clock Configuration. This bit controls the default value of the PCI Express link status register [SCC] bit. See Section 17.3.9.12, "PCI Express Link Status Register—0x5E," for more information. |

## 17.3.3 PCI Express Power Management Event and Message Registers

### 17.3.3.1 PCI Express PME and Message Detect Register (PEX_PME_MES_DR)

The PCI Express PME and message detect register, shown in Figure 17-7, logs inbound messages and PME events that are detected by the PCI Express controller. This register is a write-1-to-clear type register.

Offset 0x020 Access: w1c

| | 0–15 |
|---|---|
| R | — |
| W | |
| Reset | All zeros |

| | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23–24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | PTO | — | ENL23 | EXL23 | — | HRD | LDD | — | AION | AIB | AIOF | PION | PIB | PIOF | ABP |
| W | w1c | | w1c | w1c | | w1c | w1c | | w1c | w1c | w1c | w1c | w1c | w1c | w1c |
| Reset | All zeros | | | | | | | | | | | | | | |

Figure 17-7. PCI Express PME and Message Detect Register (PEX_PME_MES_DR)

The fields of the PCI Express PME and message detect register are described in Table 17-9.

Table 17-9. PEX_PME_MES_DR Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–15 | — | Reserved |
| 16 | PTO | PME turn off. This bit indicates the detection of a PME_Turn_Off message. This bit is only valid in EP mode.<br>1 A PME_Turn_Off_message is detected<br>0 No PME_Turn_Off message detected |
| 17 | — | Reserved. Note that during normal operation, this bit may be set (falsely). The bit may be ignored and cleared (w1c) without consequence. |

Table 17-9. PEX_PME_MES_DR Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 18 | ENL23 | Entered L2/L3 ready state. This bit indicates that the PCI Express controller has entered L2/L3 state. This is only valid in RC mode.<br>1 L2/L3 ready state has been entered<br>0 L2/L3 ready state has not been entered |
| 19 | EXL23 | Exit L2/L3 ready state. This bit indicates that the PCI Express controller has exited the L2/L3 state. This is only valid in RC mode.<br>1 Exit L2/L3 state has been detected<br>0 Exit L2/L3 state not detected |
| 20 | — | Reserved. Note that during normal operation, this bit may be set (falsely). The bit may be ignored and cleared (w1c) without consequence. |
| 21 | HRD | Hot reset detected. This bit indicates that the PCI Express controller has detected a hot reset condition on the link. The controller is reset and cleans up all outstanding transactions. Link retraining takes place once hot reset state is exited. This is valid only in EP mode.<br>1 Hot reset request has been detected<br>0 Hot reset request not detected |
| 22 | LDD | Link down detected. This bit indicates that a link down condition has been detected. The controller is reset and then cleans up all outstanding transactions. Link retraining takes place once the controller has cleaned itself up. Note that for EP, this bit and HRD are typically set when a hot reset event is detected.<br>1 Link down has been detected<br>0 Link down not detected |
| 23–24 | — | Reserved |
| 25 | AION | Attention indicator on. This bit indicates the detection of an Attention_Indicator_On message. This bit is only valid in EP mode.<br>1 Attention indicator on message is detected<br>0 No attention indicator on message detected |
| 26 | AIB | Attention indicator blink. This bit indicates the detection of an Attention_Indicator_Blink message. This bit is only valid in EP mode.<br>1 Attention indicator blink message is detected<br>0 No attention indicator blink message detected |
| 27 | AIOF | Attention indicator off. This bit indicates the detection of an Attention_Indicator_Off message. This bit is only valid in EP mode.<br>1 Attention indicator off message is detected<br>0 No attention indicator off message detected |
| 28 | PION | Power indicator on. This bit indicates the detection of a Power_Indicator_On message. This bit is only valid in EP mode.<br>1 Power indicator on message is detected<br>0 No power indicator on message detected |
| 29 | PIB | Power indicator blink. This bit indicates the detection of an Power_Indicator_Blink message. This bit is only valid in EP mode.<br>1 Power indicator blink message is detected<br>0 No power indicator blink message detected |

**Table 17-9. PEX_PME_MES_DR Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 30 | PIOF | Power indicator off. This bit indicates the detection of an Power_Indicator_Off message. This bit is only valid in EP mode.<br>1 Power indicator off message is detected<br>0 No power indicator off message detected |
| 31 | ABP | Attention button pressed. This bit indicates the detection of an Attention_Button_Pressed message. This bit is only valid in RC mode.<br>1 Attention button press message is detected<br>0 No attention button press message detected |

### 17.3.3.2 PCI Express PME and Message Disable Register (PEX_PME_MES_DISR)

The PCI Express PME and message disable register, shown in Figure 17-8, when set, prevents the detection of the corresponding bits in the PCI Express PME and message detect register.

Offset 0x024 Access: Read/Write

|  | 0–15 |
|---|---|
| R<br>W | — |
| Reset | All zeros |

|  | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23–24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | PTOD | — | ENL23D | EXL23D | — | HRDD | LDDD | — | AIOND | AIBD | AIOFD | PIOND | PIBD | PIOFD | ABPD |
| Reset | All zeros |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

**Figure 17-8. PCI Express PME and Message Disable Register (PEX_PME_MES_DISR)**

The fields of the PCI Express PME and message disable register are described in Table 17-10.

**Table 17-10. PEX_PME_MES_DISR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–15 | — | Reserved |
| 16 | PTOD | PME turn off disable. When set, this bit disables the setting of PEX_PME_MES_DR[PTO] bit.<br>1 Disable PME_Turn_Off_message detection<br>0 Enable PME_Turn_Off message detection |
| 17 | — | Reserved |
| 18 | ENL23D | Entered_L2/L3 ready disable. When set, this bit disables the setting of PEX_PME_MES_DR[ENL23] bit.<br>1 Disable Entered_L2/L3 ready state detection<br>0 Enable Entered_L2/L3 ready state detection |
| 19 | EXL23D | Exited_L2/L3 ready disable. When set, this bit disables the setting of PEX_PME_MES_DR[EXL23] bit.<br>1 Disable Exited_L2/L3 ready state detection<br>0 Enable Exited_L2/L3 ready state detection |
| 20 | — | Reserved |

Table 17-10. PEX_PME_MES_DISR Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 21 | HRDD | Hot reset detected disable. When set, this bit disables the setting of PEX_PME_MES_DR[HRD] bit.<br>1 Disable hot reset state detection<br>0 Enable hot reset state detection |
| 22 | LDDD | Link down detected disable. When set, this bit disables the setting of PEX_PME_MES_DR[LDD] bit.<br>1 Disable link down state detection<br>0 Enable link down state detection |
| 23–24 | — | Reserved |
| 25 | AIOND | Attention indicator on disable. When set, this bit disables the setting of PEX_PME_MES_DR[AION] bit.<br>1 Disable attention indicator on message detection<br>0 Enable attention indicator on message detection |
| 26 | AIBD | Attention indicator blink disable. When set, this bit disables the setting of PEX_PME_MES_DR[AIB] bit.<br>1 Disable attention indicator blink message detection<br>0 Enable attention indicator blink message detection |
| 27 | AIOFD | Attention indicator off disable. When set, this bit disables the setting of PEX_PME_MES_DR[AIOF] bit.<br>1 Disable attention indicator off message detection<br>0 Enable attention indicator off message detection |
| 28 | PIOND | Power indicator on disable. When set, this bit disables the setting of PEX_PME_MES_DR[PION] bit.<br>1 Disable power indicator on message detection<br>0 Enable power indicator on message detection |
| 29 | PIBD | Power indicator blink disable. When set, this bit disables the setting of PEX_PME_MES_DR[PIB] bit.<br>1 Disable power indicator blink message detection<br>0 Enable power indicator blink message detection |
| 30 | PIOFD | Power indicator off disable. When set, this bit disables the setting of PEX_PME_MES_DR[PIOF] bit.<br>1 Disable power indicator off message detection<br>0 Enable power indicator off message detection |
| 31 | ABPD | Attention button pressed disable. When set, this bit disables the setting of PEX_PME_MES_DR[ABP] bit.<br>1 Disable attention button press message detection<br>0 Enable attention button press message detection |

### 17.3.3.3 PCI Express PME and Message Interrupt Enable Register (PEX_PME_MES_IER)

The PCI Express PME and message interrupt enable register, shown in Figure 17-9, allows for the detection of a message or a PME event to generate an interrupt, provided that the corresponding bit in the PCI Express PME and message detect register is set.

Offset 0x028 Access: Read/Write

| | 0–15 |
|---|---|
| R<br>W | — |
| Reset | All zeros |

| | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | PTO IE | — | ENL23 IE | EXL23 IE | — | HRD IE | LDD IE | — | | AION IE | AIB IE | AIOF IE | PION IE | PIB IE | PIOF IE | ABP IE |
| Reset | All zeros | | | | | | | | | | | | | | | |

**Figure 17-9. PCI Express PME and Message Interrupt Enable Register (PEX_PME_MES_IER)**

Table 17-11 shows the fields of the PCI Express PME and message interrupt enable register.

**Table 17-11. PEX_PME_MES_IER Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–15 | — | Reserved |
| 16 | PTOIE | PME turn off interrupt enable. When set and PEX_PME_MES_DR[PTO]=1 generates an interrupt.<br>1 Enable PME_Turn_Off_message interrupt generation<br>0 Disable PME_Turn_Off message interrupt generation |
| 17 | — | Reserved |
| 18 | ENL23IE | Entered L2/L3 ready interrupt enable. When set and PEX_PME_MES_DR[ENL23]=1 generates an interrupt.<br>1 Enable Entered_L2/L3 ready state interrupt generation<br>0 Disable Entered_L2/L3 ready state interrupt generation |
| 19 | EXL23IE | Exited L2/L3 ready interrupt enable. When set and PEX_PME_MES_DR[EXL23]=1 generates an interrupt.<br>1 Enable Exited_L2/L3 ready state interrupt generation<br>0 Disable Exited_L2/L3 ready state interrupt generation |
| 20 | — | Reserved |
| 21 | HRDIE | Hot reset detected interrupt enable. When set and PEX_PME_MES_DR[HRD]=1 generates an interrupt.<br>1 Enable hot reset state interrupt generation<br>0 Disable hot reset state interrupt generation |
| 22 | LDDIE | Link down detected interrupt enable. When set and PEX_PME_MES_DR[LDD]=1 generates an interrupt.<br>1 Enable link down state interrupt generation<br>0 Disable link down state interrupt generation |
| 23–24 | — | Reserved |
| 25 | AIONIE | Attention indicator on interrupt enable. When set and PEX_PME_MES_DR[AION]=1 generates an interrupt.<br>1 Enable attention indicator on message interrupt generation<br>0 Disable attention indicator on message interrupt generation |
| 26 | AIBIE | Attention indicator blink interrupt enable. When set and PEX_PME_MES_DR[AIB]=1 generates an interrupt.<br>1 Enable attention indicator blink message interrupt generation<br>0 Disable attention indicator blink message interrupt generation |

**Table 17-11. PEX_PME_MES_IER Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 27 | AIOFIE | Attention indicator off interrupt enable. When set and PEX_PME_MES_DR[AIOF]=1 generates an interrupt.<br>1 Enable attention indicator off message interrupt generation<br>0 Disable attention indicator off message interrupt generation |
| 28 | PIONIE | Power indicator on interrupt enable. When set and PEX_PME_MES_DR[PION]=1 generates an interrupt.<br>1 Enable power indicator on message interrupt generation<br>0 Disable power indicator on message interrupt generation |
| 29 | PIBIE | Power indicator blink interrupt enable. When set and PEX_PME_MES_DR[PIB]=1 generates an interrupt.<br>1 Enable power indicator blink message interrupt generation<br>0 Disable power indicator blink message interrupt generation |
| 30 | PIOFIE | Power indicator off interrupt enable. When set and PEX_PME_MES_DR[PIOF]=1 generates an interrupt.<br>1 Enable power indicator off message interrupt generation<br>0 Disable power indicator off message interrupt generation |
| 31 | ABPIE | Attention button pressed interrupt enable. When set and PEX_PME_MES_DR[ABP]=1 generates an interrupt.<br>1 Enable attention button press message interrupt generation<br>0 Disable attention button press message interrupt generation |

### 17.3.3.4 PCI Express Power Management Command Register (PEX_PMCR)

The PCI Express power management command register, shown in Figure 17-10, provides software a mechanism to allow the PCI Express controller to get back to L0 link state.

Offset 0x02C Access: Read/Write

| | 0–28 | 29 | 30 | 31 |
|---|---|---|---|---|
| R<br>W | — | SPMES | EXL2S | PTOMR |
| Reset | All zeros | | | |

**Figure 17-10. PCI Express Power Management Command Register (PEX_PMCR)**

The fields of the PCI Express power management command register are described in Table 17-12.

**Table 17-12. PEX_PMCR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–28 | — | Reserved |
| 29 | SPMES | Set PME status. This sets the PME status bit and if PME is enabled (see Section 17.3.9.3, “PCI Express Power Management Status and Control Register—0x48,” on page 17-70 for more information) it transmits a PM_PME message upstream. This bit should not be used when in RC mode. This bit is self-clearing. |
| 30 | EXL2S | Exit L2 state. When set exits the link state out of L2/L3 ready state in order to send new requests. The request is only made when entered_L2/L3 ready state is active. This bit is self-clearing. When the link has exited L2/L3 ready state, the status bit Exit_L2/L3 ready state is set. This bit should not be used when in EP mode. |
| 31 | PTOMR | PME_Turn_Off message request. When set broadcasts a PME turn_off message. This bit should not be used when in EP mode. This bit is self-clearing |

## 17.3.4 PCI Express IP Block Revision Registers

### 17.3.4.1 IP Block Revision Register 1 (PEX_IP_BLK_REV1)

The IP block revision register 1 is shown in Figure 17-11.

Offset 0xBF8 Access: Read only

| | 0 — 15 | 16 — 23 | 24 — 31 |
|---|---|---|---|
| R | IP_ID | IP_MJ | IP_MN |
| W | | | |
| Reset | 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 | 0 0 0 0 0 0 0 1 | 0 0 0 0 0 0 0 1 |

**Figure 17-11. IP Block Revision Register 1**

Table 17-13 contains descriptions of the fields of the IP block revision register 1.

**Table 17-13. PCI Express IP Block Revision Register 1 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–15 | IP_ID | Block ID |
| 16–23 | IP_MJ | Block Major Revision |
| 24–31 | IP_MN | Block Minor Revision |

### 17.3.4.2 IP Block Revision Register 2 (PEX_IP_BLK_REV2)

The IP block revision register 2 is shown in Figure 17-12.

Offset 0xBFC Access: Read only

| | 0 — 7 | 8 — 15 | 16 — 23 | 24 — 31 |
|---|---|---|---|---|
| R | | IP_INT | | IP_CFG |
| W | | | | |
| Reset | All zeros | | | |

**Figure 17-12. IP Block Revision Register 2**

Table 17-14 contains descriptions of the fields of the IP block revision register 2.

**Table 17-14. PCI Express IP Block Revision Register 2 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–7 | — | Reserved |
| 8–15 | IP_INT | Block integration option |
| 16–23 | — | Reserved |
| 24–31 | IP_CFG | Block configuration option |

## 17.3.5 PCI Express ATMU Registers

### 17.3.5.1 PCI Express Outbound ATMU Registers

The outbound address translation windows must be aligned based on the granularity selected by the size fields. Outbound window misses use the default outbound register set (outbound ATMU window 0). Overlapping outbound windows are not supported and will cause undefined behavior. Note that for RC mode, all outbound transactions post ATMU must hit either into the memory base/limit range or the prefetchable memory base/limit range defined in the PCI Express type 1 header. For EP mode, there is no such requirement.

Note that in RC mode, there is no checking on whether the translated address actually hits into the memory base/limit range. It will just pass it through as is.

Figure 17-13 shows the outbound transaction flow.

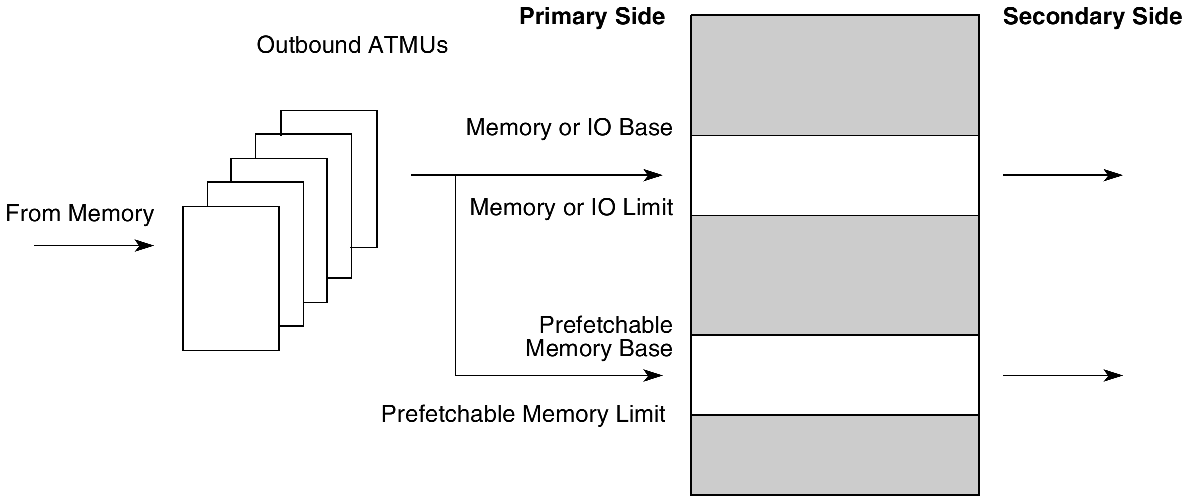


**Figure 17-13. RC Outbound Transaction Flow**

#### 17.3.5.1.1 PCI Express Outbound Translation Address Registers (PEXOTAR*n*)

The PCI Express outbound translation address registers, shown in Figure 17-14, select the starting addresses in the system address space for window hits within the PCI Express outbound address translation windows. The new translated address is created by concatenating the transaction offset to this translation address.

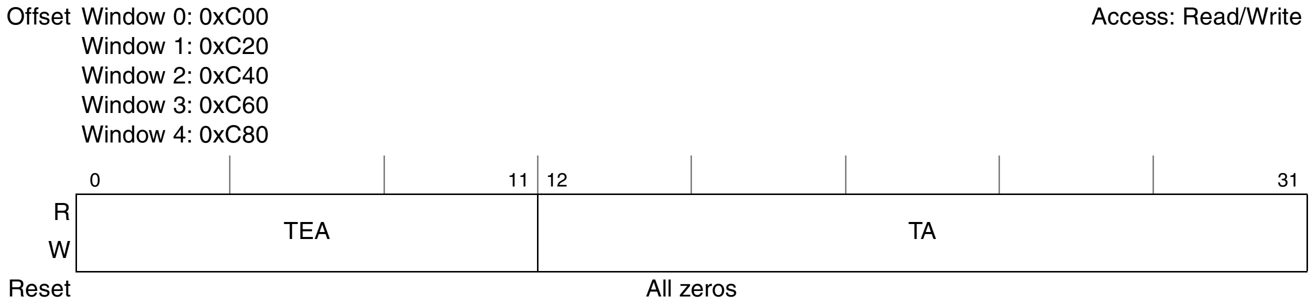


**Figure 17-14. PCI Express Outbound Translation Address Registers (PEXOTAR*n*)**

Table 17-15 describes the fields of the PCI Express outbound translation address registers.

**Table 17-15. PEXOTAR*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–11 | TEA | Translation extended address. System address which indicates the starting point of the outbound translated address. The translation address must be aligned based on the size field. Corresponds to PCI Express address bits [43:32] (bit 32 is the lsb). |
| 12–31 | TA | Translation address. System address which indicates the starting point of the outbound translated address. The translation address must be aligned based on the size field. This corresponds to PCI Express address bits [31:12]. |

### 17.3.5.1.2 PCI Express Outbound Translation Extended Address Registers (PEXOTEAR*n*)

The PCI Express outbound translation extended address registers, shown in Figure 17-15, contain the most-significant bits of a 64 bit translation address.

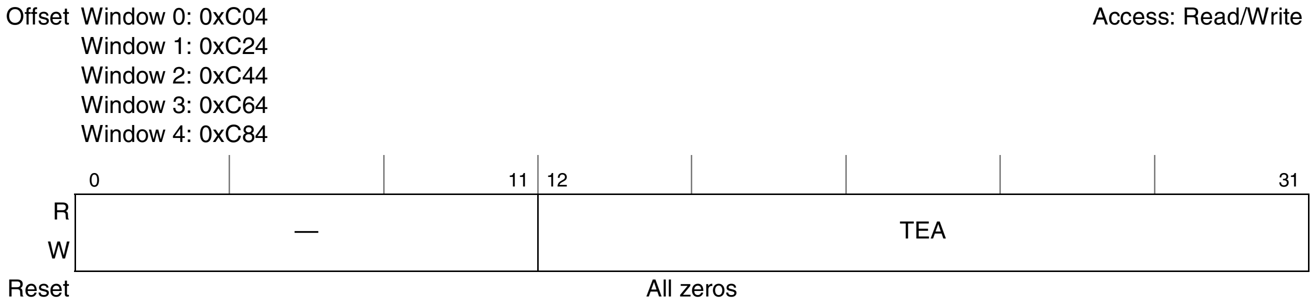


**Figure 17-15. PCI Express Outbound Translation Extended Address Registers (PEXOTEAR*n*)**

Table 17-16 describes the fields of the PCI Express outbound translation extended address registers.

**Table 17-16. PCI Express Outbound Extended Address Translation Register *n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–11 | — | Reserved |
| 12–31 | TEA | Translation extended address. System address which indicates the starting point of the outbound translated address. The translation address must be aligned based on the size field. Corresponds to PCI Express address bits [63:44]. |

#### 17.3.5.1.3 PCI Express Outbound Window Base Address Registers (PEXOWBAR*n*)

The PCI Express outbound window base address registers, shown in Figure 17-16, select the base address for the windows which are translated to the external address space. Addresses for outbound transactions are compared to these windows. If such a transaction does not fall within one of these spaces the transaction is forwarded through a default register set.

Offset Window 1: 0xC28
Window 2: 0xC48
Window 3: 0xC68
Window 4: 0xC88

Access: Read/Write

| | 0–7 | 8–11 | 12–31 |
|---|---|---|---|
| R<br>W | — | WBEA | WBA |
| Reset | All zeros | | |

**Figure 17-16. PCI Express Outbound Window Base Address Registers (PEXOWBAR*n*)**

Table 17-17 describes the fields of the PCI Express outbound window base address registers.

**Table 17-17. PCI Express Outbound Window Base Address Register *n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–7 | — | Reserved |
| 8–11 | WBEA | Window base extended address. Source address which is the starting point for the outbound translation window. The window must be aligned based on the size selected in the window size bits. Correspond to internal platform address bits [0:3]. (where 0 is the msb of the internal platform address) |
| 12–31 | WBA | Window base address. Source address which is the starting point for the outbound translation window. The window must be aligned based on the size selected in the window size bits. This corresponds to internal platform address bits [4:23]. |

#### 17.3.5.1.4 PCI Express Outbound Window Attributes Registers (PEXOWAR*n*)

The PCI Express outbound window attributes registers, shown in Figure 17-17 and Figure 17-18, define the window sizes to translate and other attributes for the translations. 64 Gbytes is the largest window size allowed. Figure 17-17 shows the outbound window attributes register 0 (PEXOWAR0).

Offset 0xC10

Access: Mixed

| | 0 | 1 | 2 | 3 | 4 | 5–7 | 8–10 | 11 | 12–15 | 16–19 | 20–25 | 26–31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | EN | — | — | ROE | NS | — | TC | — | RTT | WTT | — | OWS |
| W | | | | | | | | | | | | |
| Reset | 1 | 0 | 0 | 0 | 0 | 0 0 0 | 0 0 0 | 0 | 0 1 0 0 | 0 1 0 0 | 0 0 0 0 0 0 | 1 0 0 0 1 1 |

**Figure 17-17. PCI Express Outbound Window Attributes Register 0 (PEXOWAR0)**

Figure 17-18 shows the PCI Express outbound window attributes registers 1–4 (PEXOWAR*n*).

Offset Window 1: 0xC30
Window 2: 0xC50
Window 3: 0xC70
Window 4: 0xC90

Access: Read/Write

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | EN | — | | ROE | NS | | | | TC | | | | RTT | | | | WTT | | | | — | | | | | | OWS | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |

**Figure 17-18. PCI Express Outbound Window Attributes Registers 1–4 (PEXOWAR*n*)**

Table 17-18 describes the fields of the PCI Express outbound window attributes registers.

**Table 17-18. PEXOWAR*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | EN | Enable. This bit enables this address translation window. For the default window, this bit is read-only and always hardwired to 1.<br>0 Disable outbound translation window<br>1 Enable outbound translation window |
| 1–2 | — | Reserved |
| 3 | ROE | Relaxed ordering enable. This bit when set and the PCI Express device control register[Enable Relaxed] bit is set enables the Relaxed Ordering bit for the packet. This bit only applies to memory transactions.<br>0 Default ordering<br>1 Relaxed ordering |
| 4 | NS | No snoop enable. This bit when set and the PCI Express device control register[Enable No Snoop] bit is set enables the no snoop bit for the packet. This bit only applies to memory transactions.<br>0 Snoopable<br>1 No snoop |
| 5–7 | — | Reserved |
| 8–10 | TC | Traffic class. This field indicates the traffic class of the outbound packet. This field only applies to memory transaction. All other transaction types should set the TC field to 0.<br>000 TC0<br>001 TC1<br>010 TC2<br>011 TC3<br>100 TC4<br>101 TC5<br>110 TC6<br>111 TC7<br>**Note:** Traffic class settings are passed through to the PCI Express link, but no specific actions are taken in the device based on traffic class. |
| 11 | — | Reserved |

Table 17-18. PEXOWAR*n* Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 12–15 | RTT | Read transaction type. Read transaction type to run on the PCI Express link<br>0000 Reserved<br>0000 Reserved<br>0010 Configuration read. Supported only when in RC mode and size of less than or equal to 4 bytes and not crossing 4-byte address boundary.<br>0100 Memory read<br>... Reserved<br>1000 IO read. Supported only when in RC mode and size of less than or equal to 4 bytes and not crossing 4-byte address boundary.<br>... Reserved<br>1111 Reserved |
| 16–19 | WTT | Write transaction type. Write transaction type to run on the PCI Express link.<br>0000 Reserved<br>0001 Reserved<br>0010 Configuration write. Supported only when in RC mode and size of less than or equal to 4 bytes and not crossing 4-byte address boundary. Note that inbound write transactions on one PCI express port must not translate to outbound configuration write transactions on another PCI Express port.<br>0100 Memory write<br>0101 Message write. Only support 4-byte size access on a 4-byte address boundary.<br>... Reserved<br>1000 IO Write. Supported only when in RC mode and size of less than or equal to 4 bytes and not crossing 4-byte address boundary. Note that inbound write transactions on one PCI express port must not translate to outbound I/O write transactions on another PCI Express port.<br>... Reserved<br>1111 Reserved |
| 20–25 | — | Reserved |
| 26–31 | OWS | Outbound window size. Outbound translation window size N which is the encoded $2^{(N + 1)}$-byte window size. The smallest window size is 4 Kbytes. Note that for the default window (window 0), the outbound window size may be programmed less than the 64-Gbyte maximum. However, accesses that miss all other windows and hit outside the default window is aliased to the default window.<br>000000 Reserved<br>...<br>001011 4-Kbyte window size<br>001100 8-Kbyte window size<br>...<br>011111 4-Gbyte window size<br>100000 8-Gbyte window size<br>100001 16-Gbyte window size<br>100010 32-Gbyte window size<br>100011 64-Gbyte window size<br>100100 Reserved<br>...<br>111111 Reserved |

### 17.3.5.2 PCI Express Inbound ATMU Registers

There are differences between RC and EP implementations of inbound ATMU registers as described in the following sections.

#### 17.3.5.2.1 EP Inbound ATMU Implementation

All base address registers (BARs) reside in the PCI Express type 0 configuration header space which is accessible through the PEX_CONFIG_ADDR/PEX_CONFIG_DATA mechanism. Note that host software must program these BAR using configuration type 0 cycles. There are 4 inbound BARs.

- Default inbound window BAR0 at configuration address 0x10 (32-bit). Also known as PEXCSRBAR. This is a fixed 1-Mbyte window used for inbound memory transactions that access memory-mapped registers.
- Inbound window BAR1 at configuration address 0x14 (32-bit)
- Inbound window BAR2 at configuration address 0x18-0x1c (64-bit)
- Inbound window BAR3 at configuration address 0x20-0x24 (64-bit)

The PCI Express controller does not implement a shadow of the inbound BARs in the memory-mapped register set. However, when there is a hit to the BAR(s), the PCI Express controller uses the corresponding translation and attribute registers in the memory-mapped register set for the translation. If the transaction hits multiple BARs, then the lowest-numbered BAR is used.

#### 17.3.5.2.2 RC Inbound ATMU Implementation

In RC mode, the PEXIWBAR[1–3] registers reside outside of the type 1 header; PEXIWBAR0 is the only inbound BAR that resides in the Type 1 header (at offset 0x10).

If the transaction hits any window, the translation is performed and then the transaction is sent to memory. If there is no hit to any one of the BARs, then an UR completion is returned for non-posted transactions. All posted transactions with no BAR hit are ignored.

Figure 17-19 shows the inbound transaction flow in RC mode.

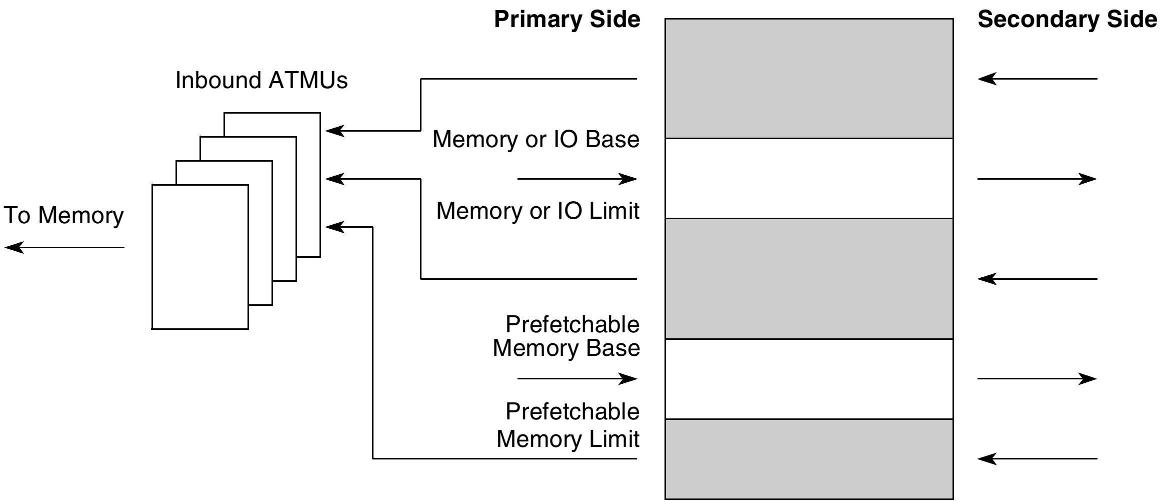


**Figure 17-19. RC Inbound Transaction Flow**

### 17.3.5.2.3 PCI Express Inbound Translation Address Registers (PEXITAR*n*)

The PCI Express inbound translation address registers, shown in Figure 17-20, contain the translated internal platform address to be used. Note that PEXITAR0 does not exist in the memory-mapped space; it is a fixed 1-Mbyte translation to the internal configuration (CCSRBAR) space.

Offset Window 1: 0xDE0
Window 2: 0xDC0
Window 3: 0xDA0

Access: Read/Write

| | 0–7 | 8–11 | 12–31 |
|---|---|---|---|
| R<br>W | — | TEA | TA |
| Reset | All zeros | | |

**Figure 17-20. PCI Express Inbound Translation Address Registers (PEXITAR*n*)**

Table 17-19 describes the fields of the PCI Express inbound translation address registers.

**Table 17-19. PCI Express Inbound Translation Address Registers Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–7 | — | Reserved |
| 8–11 | TEA | Translation extended address. Target address which indicates the starting point of the inbound translated address. The translation address must be aligned based on the size field. Corresponds to internal platform address bits [0:3] where bit 0 is the msb of the internal platform address. |
| 12–31 | TA | Translation address. Target address which indicates the starting point of the inbound translated address. The translation address must be aligned based on the size field. This corresponds to internal platform address bits [4:23]. |

### 17.3.5.2.4 PCI Express Inbound Window Base Address Registers (PEXIWBAR*n*)

The PCI Express inbound window base address registers, shown in Figure 17-21, select the base address for the windows which are translated to an alternate target address space. In root complex (RC) mode, addresses for inbound transactions are compared to these windows. In RC mode, PEXIWBAR0 is located in the PCI Express type 1 configuration header space and PEXIWBAR[1–3] registers are implemented as described in this section. In endpoint (EP) mode, these registers are not implemented in the memory-mapped space. Reading these registers in EP mode returns all zeros and writing to these offsets has no consequences. All base address registers in EP are located in the PCI Express type 0 configuration header space. Note that PEXIWBAR1 only supports 32-bit PCI Express address space.

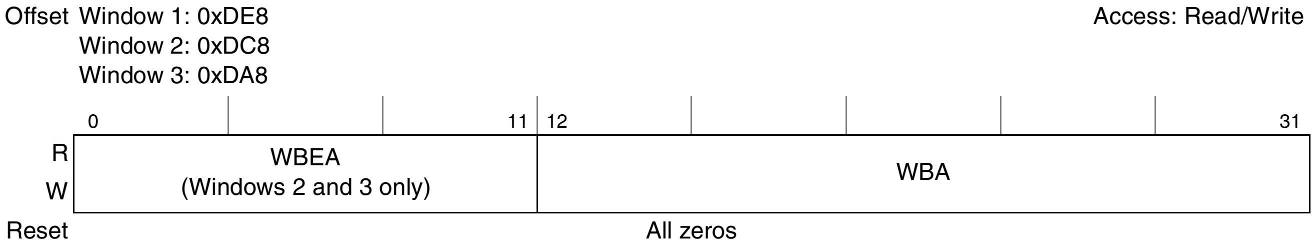


**Figure 17-21. PCI Express Inbound Window Base Address Registers (PEXIWBAR*n*)**

Table 17-20 describes the fields of the PCI Express inbound window base address registers.

**Table 17-20. PCI Express Inbound Window Base Address Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–11 | WBEA | Window base extended address. This field corresponds to PCI Express address bits [43:32]. Note that the extended address is supported for windows 2 and 3 only; for PEXIWBAR1, these bits are reserved and must be 0. |
| 12–31 | WBA | Window base address. Source address which is the starting point for the inbound translation window. The window must be aligned based on the size selected in the window size bits. This corresponds to PCI Express address bits [31:12]. |

### 17.3.5.2.5 PCI Express Inbound Window Base Extended Address Registers (PEXIWBEAR*n*)

The PCI Express inbound window base extended address registers, shown in Figure 17-22, contain the most-significant bits of a 64 bit base address.

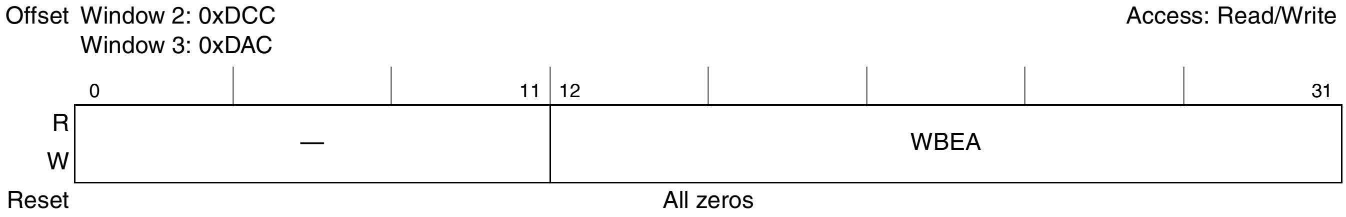


**Figure 17-22. PCI Express Inbound Window Base Extended Address Registers (PEXIWBEAR*n*)**

Table 17-21 describes the fields of the PCI Express inbound window base extended address registers.

**Table 17-21. PCI Express Inbound Window Base Extended Address Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–11 | — | Reserved |
| 12–31 | WBEA | Window base extended address. This field corresponds to PCI Express address bits [63:44] |

### 17.3.5.2.6 PCI Express Inbound Window Attributes Registers (PEXIWAR*n*)

The PCI Express inbound window attributes registers, shown in Figure 17-23, define the window sizes to translate and other attributes for the translations. 64 Gbytes is the largest window size allowed.

Offset Window 1: 0xDF0
Window 2: 0xDD0
Window 3: 0xDB0

Access: Read/Write

| | 0 | 1 | 2 | 3 7 | 8 11 | 12 15 | 16 19 | 20 25 | 26 31 |
|---|---|---|---|---|---|---|---|---|---|
| R<br>W | EN | — | PF | — | TRGT | RTT | WTT | — | IWS |
| Reset | 0 | 0 | 1 | 0 0 0 0 0 | 1 1 1 1 | 0 1 0 0 | 0 1 0 0 | 0 0 0 0 0 0 | 1 0 0 0 1 1 |

**Figure 17-23. PCI Express Inbound Window Attributes Registers (PEXIWAR*n*)**

Table 17-22 describes the fields of the PCI Express inbound window attributes registers.

**Table 17-22. PCI Express Inbound Window Attributes Registers Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | EN | Enable. This bit controls the enabling/disabling of the translation window.<br>0 Disable inbound window translation<br>1 Enable inbound window translation |
| 1 | — | Reserved |
| 2 | PF | Prefetchable. This bit indicates that the address space is prefetchable. This bit corresponds to the prefetchable bit in the BAR in the PCI Express type 0 header. This bit drives the BAR's prefetchable bit in EP mode.<br>0 Not prefetchable<br>1 Prefetchable |
| 3–7 | — | Reserved |
| 8–11 | TRGT | Target interface. If this field is set to anything other than local memory space, the attributes for the transaction must be assigned in a corresponding outbound window at the target. Values not listed below are reserved.<br>0000 PCI<br>0001 PCI Express 2—PCI Express controller 2 should not use this encoding<br>0010 PCI Express 1—PCI Express controller 1 should not use this encoding<br>0011 PCI Express 3—PCI Express controller 3 should not use this encoding<br>0100–1110 Reserved<br>1111 Local memory space<br>**Note:** Inbound write transactions on one PCI Express port must not translate to outbound configuration or I/O write transactions on another PCI Express port. |

Table 17-22. PCI Express Inbound Window Attributes Registers Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 12–15 | RTT | Read transaction type. Read transaction type to send to target interface.<br>If the transaction is not going to local memory space<br>0000 Reserved<br>...<br>0100 Read<br>0101 Reserved<br>0110 Reserved<br>0111 Reserved<br>1000 Reserved<br>...<br>1111 Reserved<br>If the transaction is going to local memory space<br>0000 Reserved<br>...<br>0100 Read, don’t snoop local processor<br>0101 Read, snoop local processor<br>0110 Reserved<br>0111 Read, snoop local processor, unlock L2 cache line<br>1000 Reserved<br>...<br>1111 Reserved |
| 16–19 | WTT | Write transaction type. Write transaction type to send to target interface.<br>If the transaction is not going to local memory space<br>0000 Reserved<br>...<br>0100 Write<br>0101 Reserved<br>0110 Reserved<br>0111 Reserved<br>1000 Reserved<br>...<br>1111 Reserved<br>If the transaction is going to local memory space<br>0000 Reserved<br>...<br>0100 Write, don’t snoop local processor<br>0101 Write, snoop local processor<br>0110 Write, snoop local processor, allocate L2 cache line<br>0111 Write, snoop local processor, allocate and lock L2 cache line<br>1000 Reserved<br>...<br>1111 Reserved |

Table 17-22. PCI Express Inbound Window Attributes Registers Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 20–25 | — | Reserved |
| 26–31 | IWS | Inbound window size. Inbound translation window size N which is the encoded $2^{(N + 1)}$-bytes window size. The smallest window size is 4 Kbytes. For EP mode, this field directly controls the size of the BARs.<br>000000 Reserved<br>...<br>001010 Reserved<br>001011 4-Kbyte window size<br>001100 8-Kbyte window size<br>...<br>011111 4-Gbyte window size<br>100000 8-Gbyte window size<br>100001 16-Gbyte window size<br>100010 32-Gbyte window size<br>100011 64-Gbyte window size<br>100100 Reserved<br>...<br>111111 Reserved |

## 17.3.6 PCI Express Error Management Registers

### 17.3.6.1 PCI Express Error Detect Register (PEX_ERR_DR)

The PCI Express error detect register, shown in Figure 17-24, contains error status bits that are detected by hardware.The detected error bits are write-1-to-clear type registers. Reading from these registers occurs normally; however, write operations can clear but not set bits. A bit is cleared whenever the register is written, and the data in the corresponding bit location is a 1. For example, to clear bit 6 and not affect any other bits in the register, the value 0b0200_0000 is written to the register. When an error is detected the appropriate error bit is set. Subsequent errors sets the appropriate error bits in the error detection registers, but only the first error for a particular unit have any relevant information captured. The interrupt enable bits are used to allow or block the error reporting to the interrupt mechanism while the disable bits are used to prevent or allow the setting of the status bits.

Offset 0xE00 Access: w1c

| | 0 | 1–7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|
| R | ME | — | PCT | — | PCAC | PNM | CDNSC | CRSNC | ICCA | IACA |
| W | w1c | | w1c | | w1c | w1c | w1c | w1c | w1c | w1c |
| Reset | All zeros | | | | | | | | | |

| | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24–31 |
|---|---|---|---|---|---|---|---|---|---|
| R | CRST | MIS | IOIS | CIS | CIEP | IOIEP | OAC | IOIA | — |
| W | w1c | w1c | w1c | w1c | w1c | w1c | w1c | w1c | |
| Reset | All zeros | | | | | | | | |

Figure 17-24. PCI Express Error Detect Register (PEX_ERR_DR)

Table 17-23 describes the fields of the PCI Express error detect register.

**Table 17-23. PCI Express Error Detect Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | ME | Multiple errors. Detecting multiple errors of the same type. An error is considered as multiple error when its detect bit is set and the same error is occurring again. Note that this bit does not track the ordering of when the error occurs.<br>1 Multiple errors were detected.<br>0 Multiple errors were not detected. |
| 1–7 | — | Reserved |
| 8 | PCT | PCI Express completion time-out. A completion time-out condition was detected for a non-posted, outbound PCI Express transaction. An error response is sent back to the requestor. Note that a completion timeout counter only starts when the non-posted request was able to send to the link partner.<br>1 A completion time-out on the PCI Express link was detected. Note that a completion timeout error is a fatal error. If a completion timeout error is detected, the system has become unstable. Hot reset is recommended to restore stability of the system.<br>0 No completion time-out on the PCI Express link detected. |
| 9 | — | Reserved |
| 10 | PCAC | PCI Express completer abort (CA) completion. A completion with CA status was received.<br>1 A completion with CA status was detected.<br>0 No completion with CA status detected. |
| 11 | PNM | PCI Express no map. Detect an inbound transaction that was not mapped to any inbound windows. In RC mode, a completion without data (Cpl) packet with a UR completion status is sent back to the requester and this bit is set. For EP mode, a Cpl packet with a UR completion status is sent back to the requester but does not set this bit.<br>1 A no-map transaction was detected in RC mode.<br>0 No no-map transaction detected. |
| 12 | CDNSC | Completion with data not successful. A completion with data packet was received with a non successful status (that is, UR, CA or CRS status).<br>1 Completion with data non successful packet was detected.<br>0 No completion with data non successful packet detected. |
| 13 | CRSNC | CRS non configuration. A completion was detected for a non configuration cycle and with CRS status.<br>1 CRS non configuration packet was detected.<br>0 No CRS non configuration packet detected. |
| 14 | ICCA | Invalid PEX_CONFIG_ADDR/PEX_CONFIG_DATA configuration access. Access to an illegal configuration space from PEX_CONFIG_ADDR/PEX_CONFIG_DATA was detected.<br>1 Invalid CONFIG_ADDR/PEX_CONFIG_DATA access detected<br>0 No invalid PEX_CONFIG_ADDR/PEX_CONFIG_DATA access detected |
| 15 | IACA | Invalid ATMU configuration access. Access to an illegal configuration space from an ATMU window was detected.<br>1 Invalid ATMU configuration access was detected<br>0 No invalid ATMU configuration access detected |
| 16 | CRST | CRS thresholded. An outbound configuration transaction was retried and thresholded due to a CRS completion status. An error response is sent back to the requestor. See Section 17.3.2.4, “PCI Express Configuration Retry Timeout Register (PEX_CONF_RTY_TOR),” for more information.<br>1 A CRS threshold condition was detected for an outbound configuration transaction<br>0 No CRS threshold condition detected |

Table 17-23. PCI Express Error Detect Register Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 17 | MIS | Message invalid size. An outbound message transaction that is greater than 4 bytes or crosses a 4-byte boundary was detected. See Section 17.4.1.9.1, “Outbound ATMU Message Generation,” for more information.<br>1 An invalid size outbound message transaction was detected<br>0 No invalid size outbound message transaction detected |
| 18 | IOIS | I/O invalid size. An outbound I/O transaction that is greater than 4 bytes or crosses a 4-byte boundary was detected.<br>1 an invalid size outbound I/O transaction was detected<br>0 no invalid size outbound I/O transaction detected |
| 19 | CIS | Configuration invalid size. An outbound configuration transaction that is greater than 4 bytes or crosses a 4-byte boundary was detected.<br>1 An invalid size outbound configuration transaction was detected<br>0 No invalid size outbound configuration transaction detected |
| 20 | CIEP | Configuration invalid EP. An outbound ATMU configuration transaction request was seen when in EP mode.<br>1 An outbound configuration transaction while in EP was detected<br>0 No outbound configuration transaction in EP detected |
| 21 | IOIEP | I/O invalid EP. An outbound I/O transaction request was seen when in EP mode.<br>1 An outbound I/O transaction while in EP was detected<br>0 No outbound I/O transaction in EP detected |
| 22 | OAC | Outbound ATMU crossing. An outbound crossing ATMU transaction was detected.<br>1 An outbound transaction that hits in one window and crosses overing it was detected<br>0 No outbound ATMU crossing condition detected |
| 23 | IOIA | I/O invalid address. An outbound I/O transaction with a translated address of greater than 4 Gbytes was detected.<br>1 A greater than 4-Gbyte I/O address was detected<br>0 No greater than 4-Gbyte I/O address detected |
| 24–31 | — | Reserved |

### 17.3.6.2 PCI Express Error Interrupt Enable Register (PEX_ERR_EN)

The PCI Express error interrupt enable register, shown in Figure 17-25, allows interrupts to be generated when the corresponding PCI Express error detect register bits are set.

Offset 0xE08 Access: Read/Write

| | 0–7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | PCT IE | — | PCAC IE | PNM IE | CDNSC IE | CRSNC IE | ICCA IE | IACA IE |
| Reset | All zeros | | | | | | | | |

| | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24–31 |
|---|---|---|---|---|---|---|---|---|---|
| R<br>W | CRST IE | MIS IE | IOIS IE | CIS IE | CIEP IE | IOIEP IE | OAC IE | IOIA IE | — |
| Reset | All zeros | | | | | | | | |

Figure 17-25. PCI Express Error Interrupt Enable Register (PEX_ERR_EN)

Table 17-24 describes the fields of the PCI Express error interrupt enable register.

**Table 17-24. PCI Express Error Interrupt Enable Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–7 | — | Reserved |
| 8 | PCTIE | PCI Express completion time-out interrupt enable. When set and PEX_ERR_DR[PCT]=1 generates an interrupt.<br>1 Enable PCI Express completion time-out interrupt generation<br>0 Disable PCI Express completion time-out interrupt generation |
| 9 | — | Reserved |
| 10 | PCACIE | PCI Express CA completion interrupt enable. When set and PEX_ERR_DR[PCAC]=1 generates an interrupt.<br>1 Enable completion with CA status interrupt generation<br>0 Disable completion with CA status interrupt generation |
| 11 | PNMIE | PCI Express no map interrupt enable. When set and PEX_ERR_DR[PNM]=1 generates an interrupt.<br>1 Enable no map PCI Express packet interrupt generation<br>0 Disable no map PCI Express packet interrupt generation |
| 12 | CDNSCIE | Completion with data not successful interrupt enable. When this bit is set and PEX_ERR_DR[CDNSC] = 1 generates an interrupt.<br>1 Enable completion with data non successful interrupt generation<br>0 Disable completion with data non successful interrupt generation |
| 13 | CRSNCIE | CRS non configuration interrupt enable. When this bit is set and PEX_ERR_DR[CRSNC] = 1 generates an interrupt.<br>1 Enable CRS non configuration interrupt generation<br>0 Disable CRS non configuration interrupt generation |
| 14 | ICCAIE | Invalid PEX_CONFIG_ADDR/PEX_CONFIG_DATA configuration access interrupt enable. When set and PEX_ERR_DR[ICCA]=1 generates an interrupt.<br>1 Enable invalid PEX_CONFIG_ADDR/PEX_CONFIG_DATA access interrupt generation<br>0 Disable invalid PEX_CONFIG_ADDR/PEX_CONFIG_DATA access interrupt generation. |
| 15 | IACAIE | Invalid ATMU configuration access. When set and PEX_ERR_DR[IACA]=1 generates an interrupt.<br>1 Enable invalid ATMU configuration access interrupt generation<br>0 Disable invalid ATMU configuration access interrupt generation |
| 16 | CRSTIE | CRS thresholded interrupt enable. When set and PEX_ERR_DR[CRST]=1 generates an interrupt.<br>1 Enable CRS threshold interrupt generation<br>0 Disable CRS threshold interrupt generation |
| 17 | MISIE | Message invalid size interrupt enable. When set and PEX_ERR_DR[MIS]=1 generates an interrupt.<br>1 Enable invalid outbound message size interrupt generation<br>0 Disable invalid outbound message size interrupt generation |
| 18 | IOISIE | I/O invalid size interrupt enable. When set and PEX_ERR_DR[IOIS]=1 generates an interrupt.<br>1 Enable invalid outbound I/O size interrupt generation<br>0 Disable invalid outbound I/O size interrupt generation |
| 19 | CISIE | Configuration invalid size interrupt enable. When set and PEX_ERR_DR[CIS]=1 generates an interrupt.<br>1 Enable invalid outbound configuration size interrupt generation<br>0 Disable invalid outbound configuration size interrupt generation |
| 20 | CIEPIE | Configuration invalid EP interrupt enable. When set and PEX_ERR_DR[CIEP]=1 generates an interrupt.<br>1 Enable outbound configuration transaction while in EP mode interrupt generation<br>0 Disable outbound configuration transaction in EP mode interrupt generation |

Table 17-24. PCI Express Error Interrupt Enable Register Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 21 | IOIEPIE | I/O invalid EP interrupt enable. When set and PEX_ERR_DR[IOIEP]=1 generates an interrupt.<br>1 Enable outbound I/O transaction EP mode interrupt generation<br>0 Disable outbound I/O transaction EP mode interrupt generation |
| 22 | OACIE | Outbound ATMU crossing interrupt enable. When set and PEX_ERR_DR[OAC]=1 generates an interrupt.<br>1 Enable outbound crossing ATMU interrupt generation<br>0 Disable outbound crossing ATMU interrupt generation |
| 23 | IOIAIE | I/O address invalid enable. When set and PEX_ERR_DR[IOIA]=1 generates an interrupt.<br>1 Enable greater than 4G I/O address interrupt generation<br>0 Disable greater than 4G I/O address interrupt generation |
| 24–31 | — | Reserved |

### 17.3.6.3 PCI Express Error Disable Register (PEX_ERR_DISR)

The PCI Express error disable register, shown in Figure 17-26, controls the setting of the PCI Express error detect register's bits.

Offset 0xE10 Access: Read/Write

| | 0 | 1–7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | MED | — | PCTD | — | PCACD | PNMD | CDNSCD | CRSNCD | ICCAD | IACAD |
| Reset | All zeros | | | | | | | | | |

| | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24–31 |
|---|---|---|---|---|---|---|---|---|---|
| R<br>W | CRSTD | MISD | IOISD | CISD | CIEPD | IOIEPD | OACD | IOIAD | — |
| Reset | All zeros | | | | | | | | |

Figure 17-26. PCI Express Error Disable Register (PEX_ERR_DISR)

Table 17-25 describes the fields of the PCI Express error disable register.

Table 17-25. PCI Express Error Disable Register Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0 | MED | Multiple errors disable. When set disables the setting of PEX_ERR_DR[ME] bit.<br>1 Disable multiple errors detection<br>0 Enable multiple errors detection |
| 1–7 | — | Reserved |
| 8 | PCTD | PCI Express completion time-out disable. When set disables the setting of PEX_ERR_DR[PCT] bit.<br>1 Disable PCI Express completion time-out detection<br>0 Enable PCI Express completion time-out detection |
| 9 | — | Reserved |
| 10 | PCACD | PCI Express CA completion disable. When set disables the setting of PEX_ERR_DR[PCAC] bit.<br>1 Disable completion with CA status detection<br>0 Enable completion with CA status detection |

### Table 17-25. PCI Express Error Disable Register Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 11 | PNMD | PCI Express no map disable. When set disables the setting of PEX_ERR_DR[PNM] bit.<br>1 Disable no map PCI Express packet detection<br>0 Enable no map PCI Express packet detection |
| 12 | CDNSCD | Completion with data not successful disable. When set disables the setting of PEX_ERR_DR[CDNSC] bit.<br>1 Disable completion with data not successful detection<br>0 Enable completion with data not successful detection |
| 13 | CRSNCD | CRS non configuration disable. When set disables the setting of PEX_ERR_DR[CRSNC] bit.<br>1 Disable CRS non configuration detection<br>0 Enable CRS non configuration detection |
| 14 | ICCAD | Invalid PEX_CONFIG_ADDR/PEX_CONFIG_DATA configuration access disable. When set disables the setting of PEX_ERR_DR[ICCA] bit.<br>1 Disable invalid PEX_CONFIG_ADDR/PEX_CONFIG_DATA access detection<br>0 Enable invalid PEX_CONFIG_ADDR/PEX_CONFIG_DATA access detection |
| 15 | IACAD | Invalid ATMU configuration access. When set disables the setting of PEX_ERR_DR[IACA] bit.<br>1 Disable invalid ATMU configuration access detection<br>0 Enable invalid ATMU configuration access detection |
| 16 | CRSTD | CRS thresholded disable. When set disables the setting of PEX_ERR_DR[CRST] bit.<br>1 Disable CRS threshold detection<br>0 Enable CRS threshold detection |
| 17 | MISD | Message invalid size disable. When set disables the setting of PEX_ERR_DR[MIS] bit.<br>1 Disable invalid outbound message size detection<br>0 Enable invalid outbound message size detection |
| 18 | IOISD | I/O invalid size disable. When set disables the setting of PEX_ERR_DR[IOIS] bit.<br>1 Disable invalid outbound I/O size detection<br>0 Enable invalid outbound I/O size detection |
| 19 | CISD | Configuration invalid size disable. When set disables the setting of PEX_ERR_DR[CIS] bit.<br>1 Disable invalid outbound configuration size detection<br>0 Enable invalid outbound configuration size detection |
| 20 | CIEPD | Configuration invalid EP disable. When set disables the setting of PEX_ERR_DR[CIEP] bit.<br>1 Disable outbound configuration transaction EP mode detection<br>0 Enable outbound configuration transaction EP mode detection |
| 21 | IOIEPD | I/O invalid EP disable. When set disables the setting of PEX_ERR_DR[IOEP] bit.<br>1 Disable outbound I/O transaction EP mode detection<br>0 Enable outbound I/O transaction EP mode detection |
| 22 | OACD | Outbound ATMU crossing disable. When set disables the setting of PEX_ERR_DR[OAC] bit.<br>1 Disable outbound crossing ATMU detection<br>0 Enable outbound crossing ATMU detection |
| 23 | IOIAD | I/O invalid address disable. When set disables the setting of PEX_ERR_DR[IOIA] bit.<br>1 Disable greater than 4G I/O address detection<br>0 Enable greater than 4G I/O address detection |
| 24–31 | — | Reserved |

### 17.3.6.4 PCI Express Error Capture Status Register (PEX_ERR_CAP_STAT)

The PCI Express error capture status register, shown in Figure 17-27, allows vital error information to be captured when an error occurs. Note that no further error capturing is performed until the ECV bit is cleared.

Offset 0xE20 Access: Mixed

| | 0–24 | 25 | 26–30 | 31 |
|---|---|---|---|---|
| R | — | TO | GSID | ECV |
| W | | | | w1c |
| Reset | All zeros | | | |

**Figure 17-27. PCI Express Error Capture Status Register (PEX_ERR_CAP_STAT)**

Table 17-26 describes the fields of the PCI Express error capture status register.

**Table 17-26. PCI Express Error Capture Status Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–24 | — | Reserved |
| 25 | TO | Transaction originator. This field Indicates whether the originator of the transaction is from PEX_CONFIG_ADDR/PEX_CONFIG_DATA.<br>1 Transaction originated from PEX_CONFIG_ADDR/PEX_CONFIG_DATA.<br>0 Transaction not originated from PEX_CONFIG_ADDR/PEX_CONFIG_DATA. |
| 26–30 | GSID | Global source ID. This field indicates the internal platform global source ID that the error transaction originates. This field only applies to non PEX_CONFIG_ADDR/PEX_CONFIG_DATA transactions.<br>00000 PCI<br>00001 PCI Express 2<br>00010 PCI Express 1<br>00011 PCI Express 3<br>00101 USB 1, USB2, or USB3<br>01101 SATA 1 or SATA 2<br>01010 Boot sequencer<br>01011 eSDHC<br>10000 Processor instruction<br>10001 Processor data<br>10101 DMA<br>11000 eTSEC1 or Security<br>11010 eTSEC3<br>All other settings reserved. |
| 31 | ECV | Error capture valid. This bit indicates that the capture registers 0-3 contain valid info. This bit when set indicates that the captured registers contain valid capturing information. No new capturing is done unless this bit is cleared by writing a 1 to it. |

### 17.3.6.5 PCI Express Error Capture Register 0 (PEX_ERR_CAP_R0)

Together with the other PCI Express error capture registers, PEX_ERR_CAP_R0 allows vital error information to be captured when an error occurs. Different error information is reported depending on whether the error source is from an outbound transaction from an internal source or from an inbound transaction from an external source; the source of the captured error is reflected in PEX_ERR_CAP_STAT[GSID]. Note that after the initial error is captured, no further capturing is performed until the PEX_ERR_CAP_STAT[ECV] bit is clear.

#### 17.3.6.5.1 PEX_ERR_CAP_R0—Outbound Case

PEX_ERR_CAP_R0 for the case when the error is caused by an outbound transaction from an internal source (that is, PEX_ERR_CAP_STAT[GSID] ≠ 0h02 ) and the error is due to timeout condition or PEX_CONFIG_ADDR/PEX_CONFIG_DATA access, is shown in Figure 17-28.

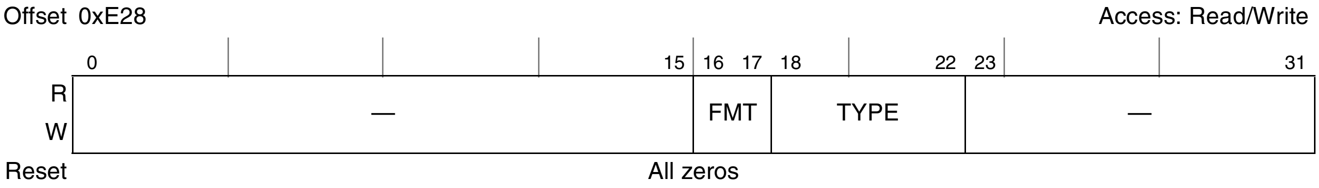


**Figure 17-28. PCI Express Error Capture Register 0 (PEX_ERR_CAP_R0) Internal Source, Outbound Transaction**

Table 17-27 describes the fields of the PCI Express error capture register 0 for the case when the error is caused by an outbound transaction from an internal source.

**Table 17-27. PCI Express Error Capture Register 0 Field Descriptions Internal Source, Outbound Transaction**

| Bits | Name | Description |
|---|---|---|
| 0–15 | — | Reserved |
| 16-17 | FMT | PCI Express format. This field indicates the PCI Express packet format. See PCI Express Spec 1.0a for more information on 3 or 4 DW (4-byte) header format. |
| 18–22 | TYPE | PCI Express type. This field indicates the PCI express packet type. See PCI Express Spec 1.0a for more information on 3 or 4 DW (4-byte) header format. |
| 23–31 | — | Reserved |

#### 17.3.6.5.2 PEX_ERR_CAP_R0—Inbound Case

PEX_ERR_CAP_R0 for the case when the error is caused by an inbound transaction from an external source (that is, PEX_ERR_CAP_STAT[GSID] = 0h02 for controller 1), is shown in Figure 17-29.

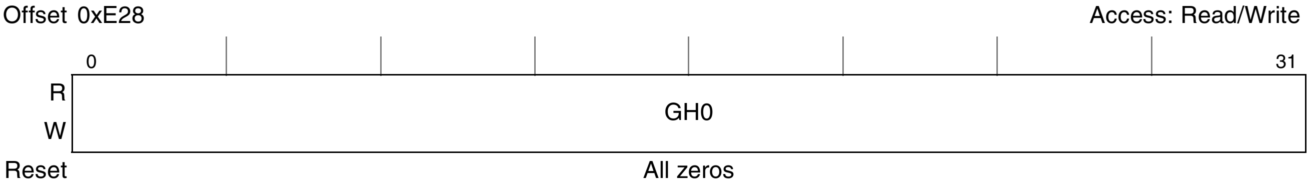


**Figure 17-29. PCI Express Error Capture Register 0 (PEX_ERR_CAP_R0) External Source, Inbound Transaction**

Table 17-28 describes the fields of PEX_ERR_CAP_R0 for the case when the error is caused by an inbound transaction from an external source.

**Table 17-28. PCI Express Error Capture Register 0 Field Descriptions External Source, Inbound Transaction**

| Bits | Name | Description |
|---|---|---|
| 0–31 | GH0 | PCI Express first DW (4-byte) header. This field contains the first DW (4-byte) of the captured PCI Express packet header.<br>27–31 TYPE<br>25–26 FMT<br>20–24 Reserved<br>17–19 TC<br>16 Reserved<br>14–15 LENGTH[9:8]<br>12–13 Reserved<br>10–11 ATTR<br>9 EP<br>8 TD<br>0–7 LENGTH[7:0] |

## 17.3.6.6 PCI Express Error Capture Register 1 (PEX_ERR_CAP_R1)

Together with the other PCI Express error capture registers, PEX_ERR_CAP_R1 allows vital error information to be captured when an error occurs. Different error information is reported depending on whether the error source is from an outbound transaction from an internal source or from an inbound transaction from an external source; the source of the captured error is reflected in PEX_ERR_CAP_STAT[GSID]. Note that after the initial error is captured, no further capturing is performed until the PEX_ERR_CAP_STAT[ECV] bit is clear.

### 17.3.6.6.1 PEX_ERR_CAP_R1—Outbound Case

PEX_ERR_CAP_R1 for the case when the error is caused by an outbound transaction from an internal source (that is, PEX_ERR_CAP_STAT[GSID] ≠ 0h02), is shown in Figure 17-30.

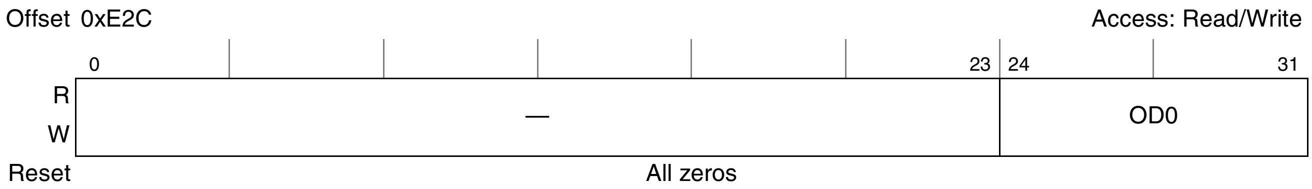


**Figure 17-30. PCI Express Error Capture Register 1 (PEX_ERR_CAP_R1) Internal Source, Outbound Transaction**

Table 17-29 describes the fields of PEX_ERR_CAP_R1 for the case when the error is caused by an outbound transaction from an internal source.

**Table 17-29. PCI Express Error Capture Register 1 Field Descriptions Internal Source, Outbound Transaction**

| Bits | Name | Description |
|---|---|---|
| 0–23 | — | Reserved |
| 24–31 | OD0 | Internal platform transaction information. Reserved for factory debug. |

### 17.3.6.6.2 PEX_ERR_CAP_R1—Inbound Case

PEX_ERR_CAP_R1 for the case when the error is caused by an inbound transaction from an external source (that is, PEX_ERR_CAP_STAT[GSID] = 0h02 for controller 1), is shown in Figure 17-31.

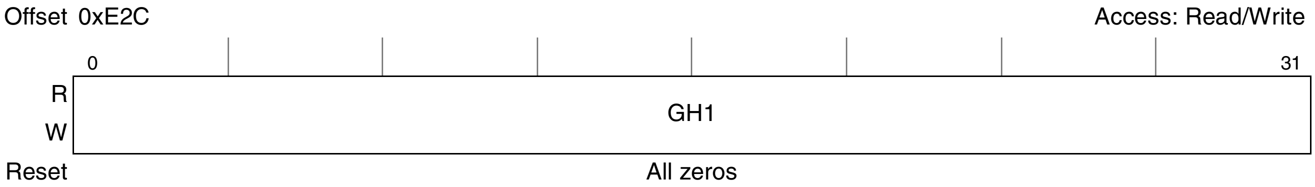


**Figure 17-31. PCI Express Error Capture Register 1 (PEX_ERR_CAP_R1) External Source, Inbound Transaction**

Table 17-30 describes the fields of PEX_ERR_CAP_R1 for the case when the FMT and TYPE subfields in PEX_ERR_CAP_R0 (see Table 17-28) indicate the error was caused by an inbound completion transaction.

**Table 17-30. PCI Express Error Capture Register 1 Field Descriptions External Source, Inbound CompletionTransaction**

| Bits | Name | Description |
|---|---|---|
| 0–31 | GH1 | PEX second DW (4-byte) header. This field contains the second DW (4-byte) of the captured PCI Express packet header.<br>24–31 Comp ID[15:8]<br>16–23 Comp ID[7:0]<br>12–15 Byte Count[11:8]<br>11 BCM<br>8–10 Comp Status<br>0–7 Byte Count[7:0] |

Table 17-31 describes the fields of PEX_ERR_CAP_R1 for the case when the FMT and TYPE subfields in PEX_ERR_CAP_R0 (see Table 17-28) indicate the error was caused by an inbound memory request transaction.

**Table 17-31. PCI Express Error Capture Register 1 Field Descriptions External Source, Inbound Memory Request Transaction**

| Bits | Name | Description |
|---|---|---|
| 0–31 | GH1 | PEX second DW (4-byte) header. This field contains the second DW (4-byte) of the captured PCI Express packet header.<br>24–31 Requester ID[15:8]<br>16–23 Requester ID[7:0]<br>8–15 Tag[7:0]<br>4–7 First DW BE[3:0]<br>0–3 Last DW BE[3:0] |

## 17.3.6.7 PCI Express Error Capture Register 2 (PEX_ERR_CAP_R2)

Together with the other PCI Express error capture registers, PEX_ERR_CAP_R2 allows vital error information to be captured when an error occurs. Different error information is reported depending on whether the error source is from an outbound transaction from an internal source or from an inbound transaction from an external source; the source of the captured error is reflected in PEX_ERR_CAP_STAT[GSID]. Note that after the initial error is captured, no further capturing is performed until the PEX_ERR_CAP_STAT[ECV] bit is clear.

### 17.3.6.7.1 PEX_ERR_CAP_R2—Outbound Case

PEX_ERR_CAP_R2 for the case when the error is caused by an outbound transaction from an internal source (that is, PEX_ERR_CAP_STAT[GSID] ≠ 0h02), is shown in Figure 17-32.

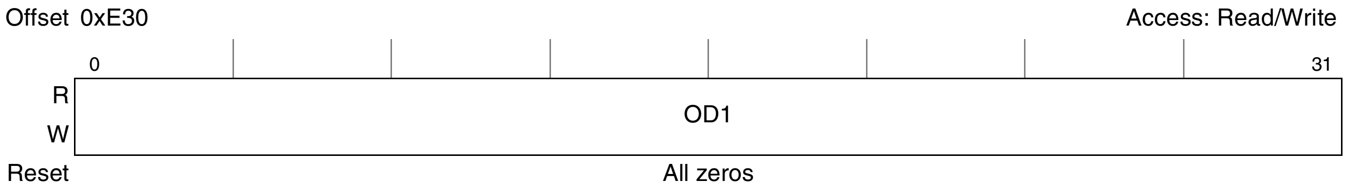


**Figure 17-32. PCI Express Error Capture Register 2 (PEX_ERR_CAP_R2) Internal Source, Outbound Transaction**

Table 17-32 describes the fields of PEX_ERR_CAP_R2 for the case when the error is caused by an outbound transaction from an internal source.

**Table 17-32. PCI Express Error Capture Register 2 Field Descriptions Internal Source, Outbound Transaction**

| Bit | Name | Description |
|---|---|---|
| 0–31 | OD1 | Internal platform transaction information. Reserved for factory debug. |

### 17.3.6.7.2 PEX_ERR_CAP_R2—Inbound Case

PEX_ERR_CAP_R2 for the case when the error is caused by an inbound transaction from an external source (that is, PEX_ERR_CAP_STAT[GSID] = 0h02 for controller 1), is shown in Figure 17-33.

Offset 0xE30 Access: Read/Write

| | 0 … 31 |
|---|---|
| R<br>W | GH2 |
| Reset | All zeros |

**Figure 17-33. PCI Express Error Capture Register 2 (PEX_ERR_CAP_R2) External Source, Inbound Transaction**

Table 17-33 describes the fields of PEX_ERR_CAP_R2 for the case when the FMT and TYPE subfields in PEX_ERR_CAP_R0 (see Table 17-28) indicate the error was caused by an inbound completion transaction.

**Table 17-33. PCI Express Error Capture Register 2 Field Descriptions External Source, Inbound Completion Transaction**

| Bits | Name | Description |
|---|---|---|
| 0–31 | GH2 | PEX third DW (4-byte) header. This field contains the third DW (4-byte) of the captured PCI Express packet header.<br>24–31 Req ID[15:8]<br>16–23 Req ID[7:0]<br>8–15 Tag[7:0]<br>1–7 Lower Address[6:0]<br>0 Reserved |

Table 17-34 describes the fields of PEX_ERR_CAP_R2 for the case when the FMT and TYPE subfields in PEX_ERR_CAP_R0 (see Table 17-28) indicate the error was caused by an inbound memory request transaction. Note that PEX_ERR_CAP_R2 captures the 32-bit address for a 3 DW memory request header or the upper half of the 64-bit address for a 4 DW memory request header; the lower half of the 64-bit address for a 4 DW memory request header is captured in PEX_ERR_CAP_R3.

**Table 17-34. PCI Express Error Capture Register 2 Field Descriptions External Source, Inbound Memory Request Transaction**

| Bits | Name | Description | |
|---|---|---|---|
| | | 3 DW Header | 4 DW Header |
| 0–31 | GH2 | PEX third DW (4-byte) header. This field contains the third DW (4-byte) of the captured PCI Express packet header. | |
| | | 24–31 Address[31:24]<br>16–23 Address[23:16]<br>8–15 Address[15:8]<br>6–7 Reserved<br>0-5 Address[7:2] | 24–31 Address[63:56]<br>16–23 Address[55:48]<br>8–15 Address[47:40]<br>0-7 Address[39:32] |

### 17.3.6.8 PCI Express Error Capture Register 3 (PEX_ERR_CAP_R3)

Together with the other PCI Express error capture registers, PEX_ERR_CAP_R3 allows vital error information to be captured when an error occurs. Different error information is reported depending on whether the error source is from an outbound transaction from an internal source or from an inbound transaction from an external source; the source of the captured error is reflected in PEX_ERR_CAP_STAT[GSID]. Note that after the initial error is captured, no further capturing is performed until the PEX_ERR_CAP_STAT[ECV] bit is clear.

#### 17.3.6.8.1 PEX_ERR_CAP_R3—Outbound Case

PEX_ERR_CAP_R3 for the case when the error is caused by an outbound transaction from an internal source (that is, PEX_ERR_CAP_STAT[GSID] ≠ 0h02), is shown in Figure 17-34.

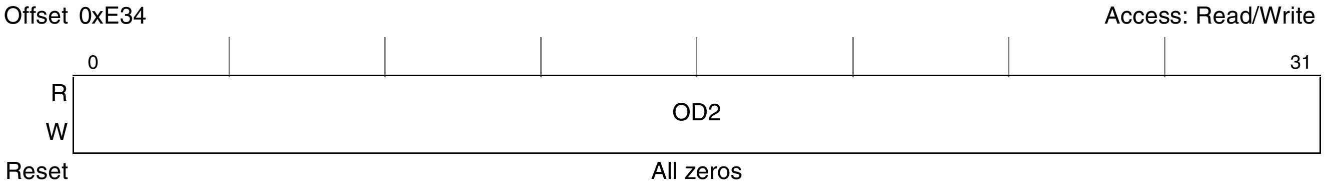


**Figure 17-34. PCI Express Error Capture Register 3 (PEX_ERR_CAP_R3) Internal Source, Outbound Transaction**

Table 17-35 describes the fields of PEX_ERR_CAP_R3 for the case when the error is caused by an outbound transaction from an internal source.

**Table 17-35. PCI Express Error Capture Register 3 Field Descriptions Internal Source, Outbound Transaction**

| Bits | Name | Description |
|---|---|---|
| 0–31 | OD2 | Internal platform transaction information. Reserved for factory debug. |

#### 17.3.6.8.2 PEX_ERR_CAP_R3—Inbound Case

PEX_ERR_CAP_R3 for the case when the error is caused by an inbound transaction from an external source (that is, PEX_ERR_CAP_STAT[GSID] = 0h02 for controller 1), is shown in Figure 17-35.

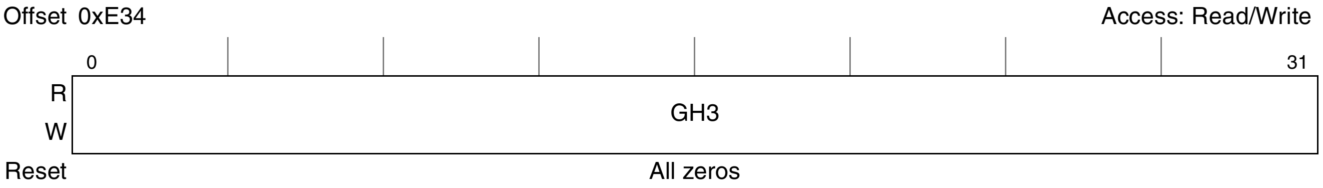


**Figure 17-35. PCI Express Error Capture Register 3 (PEX_ERR_CAP_R3) External Source, Inbound Transaction**

Table 17-36 describes the fields of PEX_ERR_CAP_R3 for the case when the FMT and TYPE subfields in PEX_ERR_CAP_R0 (see Table 17-28) indicate the error was caused by an inbound memory request transaction. Note that PEX_ERR_CAP_R3 captures the lower half of the 64-bit address for a 4 DW memory request header; the upper half of the 64-bit address for a 4 DW memory request header or the 32-bit address for a 3 DW memory request header is captured in PEX_ERR_CAP_R2.

**Table 17-36. PEX Error Capture Register 3 Field Descriptions External Source, Inbound Memory Request Transaction**

| Bits | Name | Description |
|---|---|---|
| 0–31 | GH3 | PEX fourth DW (4-byte) header. This field contains the fourth DW (4-byte) of the captured PCI Express packet header.<br>24–31 Address[31:24]<br>16–23 Address[23:16]<br>8–15 Address[15:8]<br>6–7 Reserved<br>0-5 Address[7:2] |

## 17.3.7 PCI Express Configuration Space Access

There are two methods of accessing the PCI Express configuration header:

- PCI Express outbound ATMU window
- PCI Express configuration access registers (PEX_CONFIG_ADDR/PEX_CONFIG_DATA)

### 17.3.7.1 RC Configuration Register Access

To access internal configuration space, software must rely on the PCI Express configuration access register (PEX_CONFIG_ADDR/ PEX_CONFIG_DATA) mechanism. To access external configuration space, software can either use configuration access registers or the outbound ATMU mechanism. For the configuration access register method, a value must be written to the PEX_CONFIG_ADDR register that specifies the targeted PCI Express bus, the targeted device on that bus, the targeted function within the device, and the configuration register in that device that should be accessed. The PCI Express controller's bus number is obtained from the PCI Express configuration header (type 1). Then either a write or a read to the PEX_CONFIG_DATA register triggers the actual write or read cycle to the configuration space. Note that accesses to the little-endian PCI Express configuration space must be properly formatted. See Section 17.4.1.2.1, "Byte Order for Configuration Transactions," for more information.

Note that external configuration transactions should not be attempted until the link has successfully trained. Software can poll the LTSSM state status register (PEX_LTSSM_STAT) to check the status of link training before issuing external configuration requests.

#### 17.3.7.1.1 PCI Express Configuration Access Register Mechanism

There are two types of configuration transactions (Type 0 and Type 1) needed to support hierarchical bridges.

- If the targeted bus number, and targeted device number equal to the PCI Express controller's bus number and device number, and the targeted function number is zero, then an internal PCI Express configuration cycle access is performed.

- If the targeted bus number does not equal the PCI Express controller’s bus number, but does equal the secondary bus number (from the type 1 header) and the targeted device number is 0, then a Type 0 configuration transaction is sent to the PCI Express link.
- If the targeted bus number does not equal the PCI Express controller’s bus number, and does not equal the secondary bus number (from the type 1 header), and the targeted bus number is less than or equal to the subordinate bus number (from the type 1 header), then a Type 1 configuration transaction is sent to the PCI Express link.
- If none of the above conditions occur, then the PCI Express controller returns all 1s for reads and ignores writes.

#### 17.3.7.1.2 Outbound ATMU Configuration Mechanism (RC-Only)

Software can also program one of the outbound ATMU windows to perform a configuration access. This is accomplished by programming the ReadTType or WriteTType field of the desired PEXOWAR to 0x2. Software must only issue 4-byte or less access to the ATMU configuration window and the access cannot cross a 4-byte boundary. The targeted bus number, targeted device number, targeted function number, register, and targeted extended register number sent are decoded from the outbound translated PCI Express address.

- targeted bus number[7:0] = PCI Express address[27:20]
- targeted device number[4:0] = PCI Express address[19:15]
- targeted function number[2:0] = PCI Express address[14:12]
- targeted extended register number[3:0] = PCI Express address[11:8]
- targeted register number[5:0] = PCI Express address[7:2]

A Type 0 configuration cycle is sent to the link if the targeted bus number equals the secondary bus number (from the type 1 header) and targeted device number is 0. A Type 1 configuration cycle is sent to the link if targeted bus number does not equal primary bus and secondary bus numbers and it is less than or equal to the subordinate bus number (from the type 1 header). For all other cases, the PCI Express controller squashes the write and read results in a response with error returned.

Note that the PCI Express controller does not support access to its internal configuration registers using the outbound ATMU mechanism. That is, the outbound ATMU mechanism must not be used to program the internal registers.

### 17.3.7.2 EP Configuration Register Access

When the PCI Express controller is configured as an EP device it responds to remote host generated configuration cycles. This is indicated by decoding the configuration command along with type 0 access in the packet. A remote host can access all of the PCI Express configuration area except the PCI Express Controller Internal CSR registers in the extended PCI Express configuration space at offsets 0x400–0x6FF. The PCI Express Controller Internal CSR registers are not accessible by inbound PCI Express configuration transactions. Attempts to access these registers return all zeros.

While in EP mode, the PCI Express controller does not support generating configuration accesses as a master. All accesses to PEX_CONFIG_ADDR/PEX_CONFIG_DATA cause the device to access the internal configuration registers regardless of the targeted bus number or targeted device number programmed in the PEX_CONFIG_ADDR register. There is no configuration mechanism supported in EP mode using the ATMU window. If the outbound ATMU window is configured to issue a configuration transaction, all posted transactions hitting this window are ignored and all non-posted transactions get a response with an error.

## 17.3.8 PCI Compatible Configuration Headers

The first 64 bytes of the 256-byte PCI compatible configuration space consists of a predefined header that every PCI device must support. The first 16 bytes of the predefined header are defined the same for all PCI Express devices. These common registers are shown in Figure 17-36.

▨ Reserved

| | | | | Address Offset (Hex) |
|---|---|---|---|---|
| Device ID | | Vendor ID | | 00 |
| Status | | Command | | 04 |
| Class Code | | | Revision ID | 08 |
| BIST | Header Type | Latency Timer | Cache Line Size | 0C |

**Figure 17-36. PCI Express PCI-Compatible Configuration Header Common Registers**

The remaining 48 bytes of the header may have differing layouts depending on the function of the device. There are two header types applicable to PCI Express. Type 0 headers are typically used by endpoints; Type 1 headers are used by root complexes and switches/bridges.

### 17.3.8.1 Common PCI Compatible Configuration Header Registers

This section details the registers that are common to both type 0 and type 1 configuration headers.

#### 17.3.8.1.1 PCI Express Vendor ID Register—Offset 0x00

The vendor ID register, shown in Figure 17-37, is used to identify the manufacturer of the device.

Offset 0x00 Access: Read only

| | 15 | | | | | | | | | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Vendor ID | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 |

**Figure 17-37. PCI Express Vendor ID Register**

Table 17-37 describes the vendor ID register fields.

**Table 17-37. PCI Express Vendor ID Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 15–0 | Vendor ID | 0x1957 (Freescale) |

#### 17.3.8.1.2 PCI Express Device ID Register—Offset 0x02

The device ID register, shown in Figure 17-38, is used to identify the device.

Offset 0x02 Access: Read only

| | 15 … 0 |
|---|---|
| R | Device ID |
| W | |
| Reset | Device-specific; see field description |

**Figure 17-38. PCI Express Device ID Register**

Table 17-38 describes the device ID register fields.

**Table 17-38. PCI Express Device ID Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 15–0 | Device ID | 0x0050 MPC8536E<br>0x0051 MPC8536 |

### 17.3.8.1.3 PCI Express Command Register—Offset 0x04

The command register, shown in Figure 17-39, provides control over the ability to generate and respond to PCI Express cycles.

Offset 0x04 Access: Mixed

| | 15–11 | 10 | 9 | 8 | 7 | 6 | 5–3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| R | — | Interrupt Disable | — | SERR | — | Parity error response | — | Bus master | Memory space | I/O space |
| W | | | | | | | | | | |
| Reset | All zeros | | | | | | | | | |

**Figure 17-39. PCI Express Command Register**

Table 17-39 describes the bits of the command register.

**Table 17-39. PCI Express Command Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 15–11 | — | Reserved |
| 10 | Interrupt Disable | Controls the ability to generate INTx interrupt messages.<br>0 Enables INTx interrupt messages<br>1 Disables INTx interrupt messages<br>Any INTx emulation interrupts already asserted by this device must be deasserted when this bit is set. |
| 9 | — | Reserved |
| 8 | SERR | Controls the reporting of fatal and non-fatal errors detected by the device to the root complex.<br>0 Disables reporting<br>1 Enables reporting<br>**Note:** The error control and status bits in the command and status registers control PCI-compatible error reporting. PCI Express advanced error reporting is controlled by the PCI Express device control register described in Section 17.3.9.8, “PCI Express Device Control Register—0x54,” and the advance error reporting capability structure described in sections 17.3.10.1 through 17.3.10.12. |
| 7 | — | Reserved |

Table 17-39. PCI Express Command Register Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 6 | Parity error response | Controls whether this PCI Express controller responds to parity errors.<br>0 Parity errors are ignored and normal operation continues.<br>1 Parity errors cause the appropriate bit in the PCI Express status register to be set. However, note that errors are reported based on the values set in the PCI Express error enable and detection registers.<br>**Note:** The error control and status bits in the command and status registers control PCI-compatible error reporting. PCI Express advanced error reporting is controlled by the PCI Express device control register described in Section 17.3.9.8, "PCI Express Device Control Register—0x54," and the advance error reporting capability structure described in sections 17.3.10.1 through 17.3.10.12. |
| 5–3 | — | Reserved |
| 2 | Bus master | Indicates whether this PCI Express device is configured as a master.<br>0 Disables the ability to generate PCI Express accesses<br>1 Enables this PCI Express controller to behave as a PCI Express bus master<br>EP mode: Clearing this bit prevent the device from issuing any memory or I/O transactions. Because MSI interrupts are effectively memory writes, clearing this bit also disables the ability of the device to issue MSI interrupts.<br>RC mode: Clearing this bit disables the ability of the device to forward memory transactions upstream. This causes any inbound memory transaction to be treated as an unsupported request. |
| 1 | Memory space | Controls whether this PCI Express device (as a target) responds to memory accesses.<br>0 This PCI Express device does not respond to PCI Express memory space accesses.<br>1 This PCI Express device responds to PCI Express memory space accesses.<br>EP mode: Clearing this bit prevents the device from accepting any memory transaction.<br>RC mode: This bit is ignored. It does not affect outbound memory transaction |
| 0 | I/O space | I/O space.<br>0 This PCI Express device (as a target) does not respond to PCI Express I/O space accesses.<br>1 This PCI Express device (as a target) does respond to PCI Express I/O space accesses.<br>EP mode: Clearing this bit prevents the device from accepting any IO transaction. Note that this bit is a don't care in EP mode since the device does not support IO transaction.<br>RC mode: This bit is ignored. It does not affect outbound IO transaction. |

#### 17.3.8.1.4 PCI Express Status Register—Offset 0x06

The status register, shown in Figure 17-40, is used to record status information for PCI Express related events.

Offset 0x06 Access: Mixed

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| R | Detected parity error | Signaled system error | Received master-abort | Received target-abort | Signaled target-abort | — | | Master data parity error |
| W | w1c | w1c | w1c | w1c | w1c | | | w1c |
| Reset | All zeros | | | | | | | |

| | 7 | | 5 | 4 | 3 | 2 | | 0 |
|---|---|---|---|---|---|---|---|---|
| R | — | | | Capabilities list | Interrupt Status | — | | |
| W | | | | | | | | |
| Reset | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

Figure 17-40. PCI Express Status Register

The definition of each bit is given in Table 17-40.

**Table 17-40. PCI Express Status Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 15 | Detected parity error[1] | Set whenever a device receives a poisoned TLP regardless of the state of bit 6 in the command register. |
| 14 | Signaled system error[1] | Set whenever a device sends a ERR_FATAL or ERR_NONFATAL message and the SERR enable bit in the command register is set. |
| 13 | Received master-abort[1] | Set whenever a requestor receives a completion with unsupported request completion status. |
| 12 | Received target-abort[1] | Set whenever a device receives a completion with completer abort completion status. |
| 11 | Signaled target-abort[1] | Set whenever a device completes a request using completer abort completion status. |
| 10–9 | — | Reserved |
| 8 | Master data parity error detected[1] | Set by the requestor (primary side for Type1 headers) when either the requestor receives a completion marked poisoned or the requestor poisons a write request. Note that the parity error enable bit (bit 6) in the command register must be set for this bit to be set. |
| 7–5 | — | Reserved |
| 4 | Capabilities List | All PCI Express devices are required to implement the PCI Express capability structure. |
| 3 | Interrupt Status | Set when an INTx interrupt message is pending internally to the device.<br>Note that this bit is associated with INTx messages and not Message Signaled Interrupts. |
| 2–0 | — | Reserved |

[1] The error control and status bits in the command and status registers control PCI-compatible error reporting. PCI Express advanced error reporting is controlled by the PCI Express device control register described in Section 17.3.9.8, "PCI Express Device Control Register—0x54," and the advance error reporting capability structure described in sections 17.3.10.1 through 17.3.10.12.

### 17.3.8.1.5 PCI Express Revision ID Register—Offset 0x08

The revision ID register, shown in Figure 17-41, is used to identify the revision of the device.

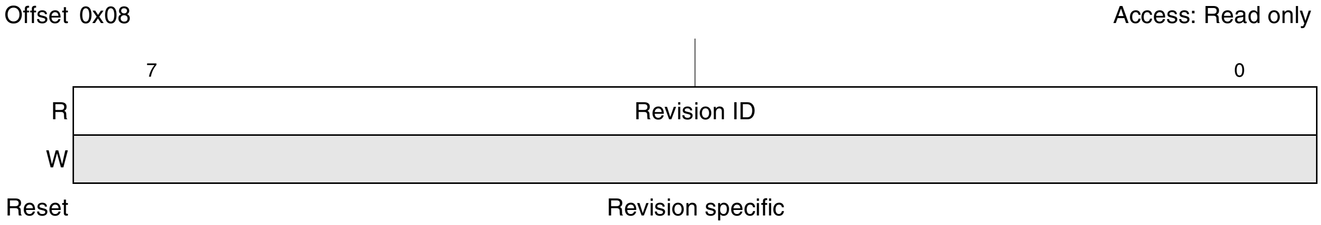


**Figure 17-41. PCI Express Revision ID Register**

Table 17-41 describes the revision ID register fields.

**Table 17-41. PCI Express Revision ID Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 7–0 | Revision ID | Revision specific. |

#### 17.3.8.1.6 PCI Express Class Code Register—Offset 0x09

The class code register, shown in Figure 17-42, is comprised of three single-byte fields—base class (offset 0x0B), sub-class (offset 0x0A), and programming interface (offset 0x09)—that indicate the basic functionality of the function.

Offset 0x09 Access: Read only

| | 23 | | | | | | | 16 | 15 | | | | | | | 8 | 7 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Base Class | | | | | | | | Sub-Class | | | | | | | | Programming Interface | |
| W | | | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0x00 RC mode; 0x01 EP mode | |

**Figure 17-42. PCI Express Class Code Register**

Table 17-42 describes the class code register fields.

**Table 17-42. PCI Express Class Code Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 23–16 | Base Class | 0x0B—Processor |
| 15–8 | Sub-Class | 0x20—PowerPC |
| 7–0 | Programming Interface | 0x00—RC mode<br>0x01—EP mode |

#### 17.3.8.1.7 PCI Express Cache Line Size Register—Offset 0x0C

The cache line size register, shown in Figure 17-43, is provided for legacy compatibility purposes (PCI 2.3); it is not used for PCI Express device functionality.

Offset 0x0C Access: Read/Write

| | 7 | 0 |
|---|---|---|
| R<br>W | Cache Line Size | |
| Reset | All zeros | |

**Figure 17-43. PCI Express Bus Cache Line Size Register**

Table 17-43 describes the cache line size register.

**Table 17-43. PCI Express Bus Cache Line Size Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 7–0 | Cache Line Size | Represents the cache line size of the processor in terms of 32-bit words (8 32-bit words = 32 bytes). Note that for PCI Express operation this register is ignored. |

### 17.3.8.1.8 PCI Express Latency Timer Register—0x0D

The latency timer register, shown in Figure 17-44, is provided for legacy compatibility purposes (PCI 2.3); it is not used for PCI Express device functionality.

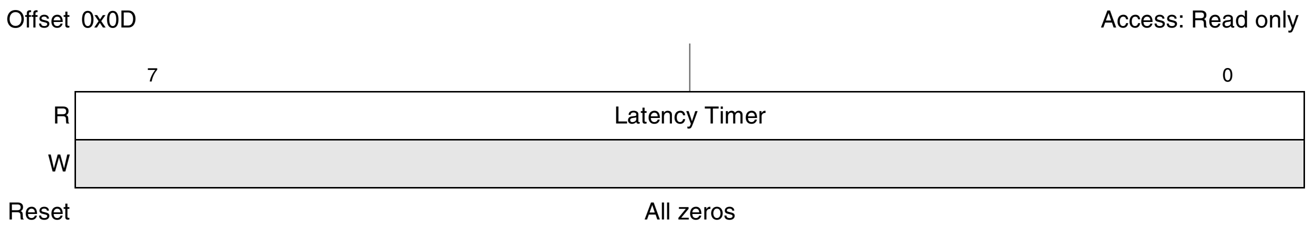


**Figure 17-44. PCI Express Bus Latency Timer Register**

Table 17-44 describes the PCI Express latency timer register (PLTR).

**Table 17-44. PCI Express Bus Latency Timer Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 7–0 | Latency Timer | Note that for PCI Express operation this register is ignored. |

### 17.3.8.1.9 PCI Express Header Type Register—0x0E

The PCI Express header type register, shown in Figure 17-43, is used to identify the layout of the PCI compatible header.

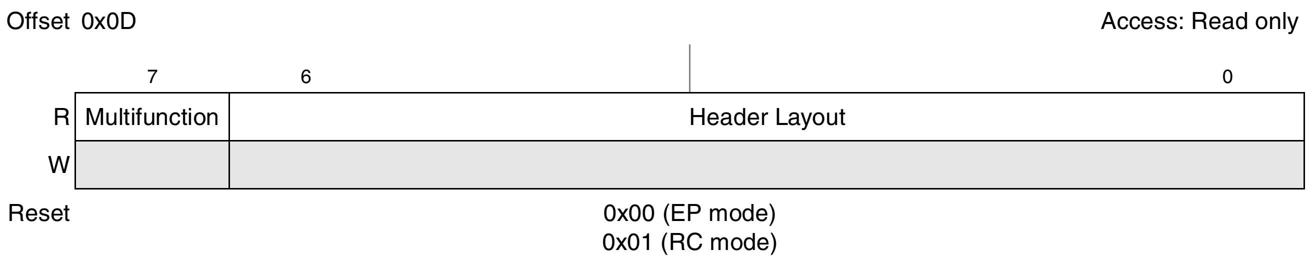


**Figure 17-45. PCI Express Bus Latency Timer Register**

Table 17-44 describes the PCI Express header type register.

**Table 17-45. PCI Express Bus Latency Timer Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 7 | Multifunction | Identifies whether a device supports multiple functions<br>0 Single function device<br>1 Multiple function device |
| 6–0 | Header Layout | 0x00 Endpoint. See Figure 17-46 for type 0 layout.<br>0x01 Root Complex. See Figure 17-58 for type 1 layout.<br>All other encodings reserved. |

#### 17.3.8.1.10 PCI Express BIST Register—0x0F

The BIST register is optional and reserved on the PCI Express controller.

### 17.3.8.2 Type 0 Configuration Header

The type 0 header is shown in Figure 17-46.

▨ Reserved

| | | | | Address Offset (Hex) |
|---|---|---|---|---|
| Device ID | | Vendor ID | | 00 |
| Status | | Command | | 04 |
| Class Code | | | Revision ID | 08 |
| BIST | Header Type | Latency Timer | Cache Line Size | 0C |
| Base Address Registers | | | | 10 |
| | | | | 14 |
| | | | | 18 |
| | | | | 1C |
| | | | | 20 |
| | | | | 24 |
| ▨ | | | | 28 |
| Subsystem ID | | Subsystem Vendor ID | | 2C |
| ▨ | | | | 30 |
| ▨ | | | Capabilities Pointer | 34 |
| Expansion ROM Base Address | | | | 38 |
| MAX_LAT | MIN_GNT | Interrupt Pin | Interrupt Line | 3C |

**Figure 17-46. PCI Express PCI-Compatible Configuration Header—Type 0**

Section 17.3.8.1, “Common PCI Compatible Configuration Header Registers,” describes the registers in the first 16 bytes of the header. This section describes the registers that are unique to the type 0 header beginning at offset 0x10.

#### 17.3.8.2.1 PCI Express Base Address Registers—0x10–0x27

The PCI Express base address registers (BARs) point to the beginning of distinct address ranges which the device should claim. In EP mode, the device supports a configuration space BAR, a 32-bit memory space BAR, and two 64-bit memory space BARs. In RC mode, the device only supports the configuration space BAR in the header; the other memory spaces are defined by the inbound ATMUs. Refer to Section 17.3.5.2, “PCI Express Inbound ATMU Registers,” for more information.

Base address register 0 at offset 0x10 is a special fixed 1-Mbyte window that is used for inbound configuration accesses. This window is called the PCI Express configuration and status register base address register (PEXCSRBAR). Note that PEXCSRBAR cannot be updated through the inbound ATMU registers. The PEXCSRBAR is shown in Figure 17-47.

Offset 0x10 Access: Mixed

| | 31–20 | 19–4 | 3 | 2–1 | 0 |
|---|---|---|---|---|---|
| R | ADDRESS | — | PREF | TYPE | MemSp |
| W | | | | | |
| Reset | All zeros | | | | |

**Figure 17-47. PCI Express Base Address Register 0 (PEXCSRBAR)**

Table 17-46 describes the PCI Express configuration and status register base address register.

**Table 17-46. PEXCSRBAR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31–20 | ADDRESS | Indicates the base address that the inbound configuration window occupies. This window is fixed at 1 Mbyte. |
| 19–4 | — | Reserved |
| 3 | PREF | Prefetchable |
| 2–1 | TYPE | Type.<br>00 Locate anywhere in 32-bit address space. |
| 0 | MemSp | Memory space indicator |

Base address register 1 at offset 0x14 is used to define the inbound memory window in the 32-bit memory space. The 32-bit memory BAR is shown in Figure 17-48.

Offset 0x14 (EP-mode only) Access: Mixed

| | 31–12 | 11–4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|
| R | ADDRESS | — | PREF | TYPE | | MemSp |
| W | | | | | | |
| Reset | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 1 | 0 | 0 | 0 |

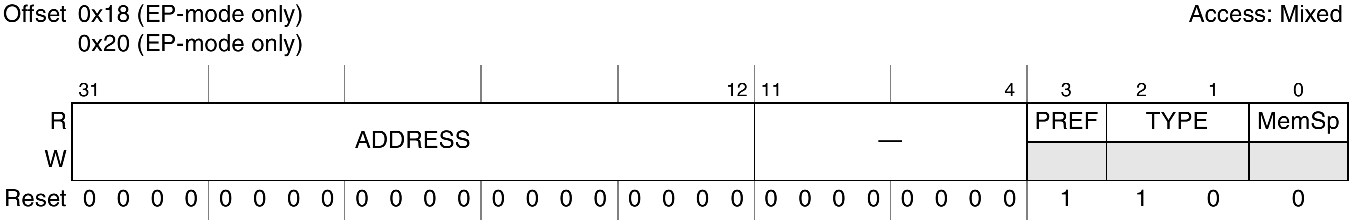
**Figure 17-48. 32-Bit Memory Base Address Register (BAR1)**

Table 17-47 describes the PCI Express 32-bit memory BAR fields.

**Table 17-47. 32-Bit Memory Base Address Register (BAR1) Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31–12 | ADDRESS | Indicates the base address where the inbound memory window begins. The number of upper bits that the device allows to be writable is selected through the inbound window size in the inbound window attributes register (PEXIWAR1). |
| 11–4 | — | Reserved. The device allows a 4 Kbyte window minimum. |
| 3 | PREF | Prefetchable. This bit is determined by PEXIWAR1[PF]. |
| 2–1 | TYPE | Type.<br>00 Locate anywhere in 32-bit address space. |
| 0 | MemSp | Memory space indicator. |

Base address register 2 at offset 0x18 and base address register 4 at offset 0x20 are used to define the lower portion of the 64-bit inbound memory windows. The 64-bit low memory BARs are shown in Figure 17-49.



Offset 0x18 (EP-mode only)
0x20 (EP-mode only)

Access: Mixed

| | 31–12 | 11–4 | 3 | 2–1 | 0 |
|---|---|---|---|---|---|
| R<br>W | ADDRESS | — | PREF | TYPE | MemSp |
| Reset | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 1 | 1 0 | 0 |

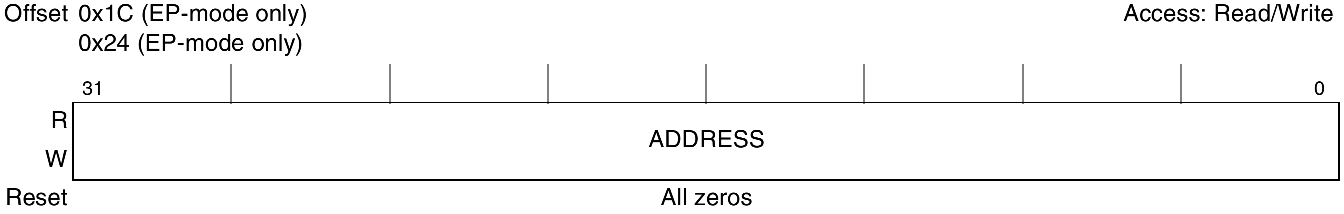
**Figure 17-49. 64-Bit Low Memory Base Address Register**

Table 17-48 describes the PCI Express 64-bit low memory BAR fields.

**Table 17-48. 64-Bit Low Memory Base Address Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31–12 | ADDRESS | Indicates the lower portion of the base address where the inbound memory window begins. The number of bits that the device allows to be writable is selected through the inbound window size in the inbound window attributes registers (PEXIWAR2 for offset 0x18 and PEXIWAR3 for offset 0x20). |
| 11–4 | — | Reserved. The device allows a 4 Kbyte window minimum. |
| 3 | PREF | Prefetchable. This bit is determined by PEXIWAR*n*[2]. |
| 2–1 | TYPE | Type.<br>0b10 Locate anywhere in 64-bit address space. |
| 0 | MemSp | Memory space indicator |

Base address register 3 at offset 0x1C and base address register 5 at offset 0x24 are used to define the upper portion of the 64-bit inbound memory windows. The 64-bit high memory BARs are shown in Figure 17-50.



Offset 0x1C (EP-mode only)
0x24 (EP-mode only)

Access: Read/Write

| | 31–0 |
|---|---|
| R<br>W | ADDRESS |
| Reset | All zeros |

**Figure 17-50. 64-Bit High Memory Base Address Register**

Table 17-49 describes the PCI Express 64-bit low memory BAR fields.

**Table 17-49. Bit Setting for 64-Bit High Memory Base Address Register**

| Bits | Name | Description |
|---|---|---|
| 31–0 | ADDRESS | Indicates the upper portion of the base address where the inbound memory window begins. The number of bits that the device allows to be writable is selected through the inbound window size in the inbound window attributes registers (PEXIWAR2 for offset 0x1C and PEXIWAR3 for offset 0x24). If no access to local memory is to be permitted by external requestors, then all bits are programmed. |

### 17.3.8.2.2 PCI Express Subsystem Vendor ID Register (EP-Mode Only)—0x2C

The PCI Express subsystem vendor ID register is used to identify the subsystem.

Offset 0x2C (EP-mode only) Access: Read only

| | 15 … 0 |
|---|---|
| R | Subsystem Vendor ID |
| W | |
| Reset | Value taken from PEX_SSVID_UPDATE[SSV_ID] |

**Figure 17-51. PCI Express Subsystem Vendor ID Register**

**Table 17-50. PCI Express Subsystem Vendor ID Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 15–0 | Subsystem Vendor ID | The value for subsystem vendor ID is determined by the PCI Express subsystem vendor ID update register. See Section 17.3.10.17, "PCI Express Subsystem Vendor ID Update Register (EP Mode Only)—0x478," for more information. |

### 17.3.8.2.3 PCI Express Subsystem ID Register (EP-Mode Only)—0x2E

The PCI Express subsystem ID register is used to identify the subsystem.

Offset 0x2E (EP-mode only) Access: Read only

| | 15 … 0 |
|---|---|
| R | Subsystem ID |
| W | |
| Reset | Value taken from PEX_SSVID_UPDATE[SSV_ID] |

**Figure 17-52. PCI Express Subsystem ID Register**

**Table 17-51. PCI Express Subsystem ID Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 15–0 | Subsystem ID | The value for subsystem ID is determined by the PCI Express subsystem vendor ID update register. See Section 17.3.10.17, "PCI Express Subsystem Vendor ID Update Register (EP Mode Only)—0x478," for more information. |

### 17.3.8.2.4 Capabilities Pointer Register—0x34

The capabilities pointer identifies additional functionality supported by the device.

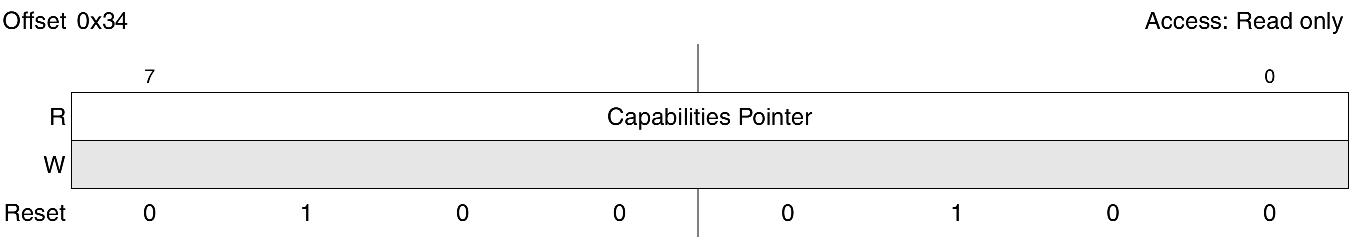


**Figure 17-53. Capabilities Pointer Register**

**Table 17-52. Capabilities Pointer Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 7–0 | Capabilities Pointer | The capabilities pointer provides the offset (0x44) for additional PCI-compatible registers above the common 64-byte header. Refer to Section 17.3.9, "PCI Compatible Device-Specific Configuration Space," for more information. |

### 17.3.8.2.5 PCI Express Interrupt Line Register (EP-Mode Only)—0x3C

The interrupt line register is used by device drivers and OS software to communicate interrupt line routing information. Values in this register are programmed by system software and are system specific.

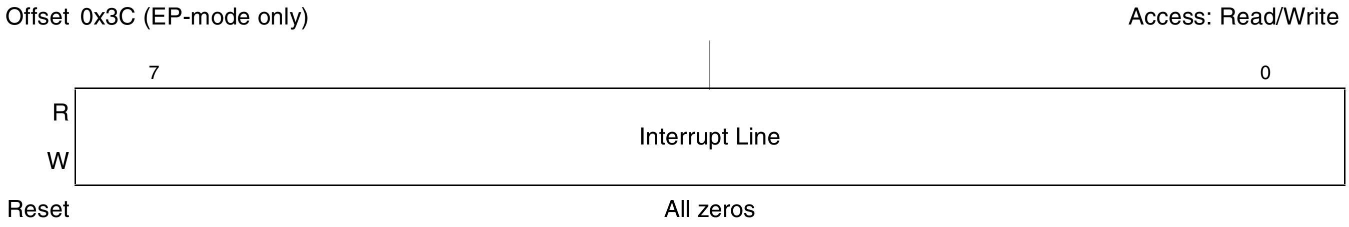


**Figure 17-54. PCI Express Interrupt Line Register**

**Table 17-53. PCI Express Interrupt Line Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 7–0 | Interrupt Line | Used to communicate interrupt line routing information. |

### 17.3.8.2.6 PCI Express Interrupt Pin Register—0x3D

The interrupt pin register identifies the legacy interrupt (INTx) messages the device (or function) uses.

Offset 0x3D Access: Read only

| | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|
| R | Interrupt Pin | | | | | | | |
| W | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

Figure 17-55. PCI Express Interrupt Pin Register

Table 17-54. PCI Express Interrupt Pin Register Field Description

| Bits | Name | Description |
|---|---|---|
| 7–0 | Interrupt pin | Legacy INTx message used by this device.<br>0x00 This device does not use legacy interrupt (INTx) messages.<br>0x01 INTA<br>0x02 INTB<br>0x03 INTC<br>0x04 INTD<br>all others Reserved. |

### 17.3.8.2.7 PCI Express Minimum Grant Register (EP-Mode Only)—0x3E

This register does not apply to PCI Express. It is present for legacy purposes.

Offset 0x3E (EP-mode only) Access: Read only

| | 7 | 0 |
|---|---|---|
| R | MIN_GNT | |
| W | | |
| Reset | All zeros | |

Figure 17-56. PCI Express Maximum Grant Register (MAX_GNT)

Table 17-55. PCI Express Maximum Grant Register Field Description

| Bits | Name | Description |
|---|---|---|
| 7–0 | MIN_GNT | Does not apply for PCI Express. |

#### 17.3.8.2.8 PCI Express Maximum Latency Register (EP-Mode Only)—0x3F

This register does not apply to PCI Express. It is present for legacy purposes.

Offset 0x3F (EP-mode only) Access: Read only

| | 7 | 0 |
|---|---|---|
| R | MAX_LAT | |
| W | ▨ | |
| Reset | All zeros | |

Figure 17-57. PCI Express Maximum Latency Register (MAX_LAT)

Table 17-56. PCI Express Maximum Latency Register Field Description

| Bits | Name | Description |
|---|---|---|
| 7–0 | MAX_LAT | Does not apply for PCI Express. |

### 17.3.8.3 Type 1 Configuration Header

The type 1 header is shown in Figure 17-58.

▨ Reserved

| | | | | Address Offset (Hex) |
|---|---|---|---|---|
| Device ID | | Vendor ID | | 00 |
| Status | | Command | | 04 |
| Class Code | | | Revision ID | 08 |
| BIST | Header Type | Latency Timer | Cache Line Size | 0C |
| Base Address Register 0 | | | | 10 |
| ▨ | | | | 14 |
| Secondary Latency Timer | Subordinate Bus Number | Secondary Bus Number | Primary Bus Number | 18 |
| Secondary Status | | I/O Limit | I/O Base | 1C |
| Memory Limit | | Memory Base | | 20 |
| Prefetchable Memory Limit | | Prefetchable Memory Base | | 24 |
| Prefetchable Base Upper 32 Bits | | | | 28 |
| Prefetchable Limit Upper 32 Bits | | | | 2C |
| I/O Limit Upper 16 Bits | | I/O Base Upper 16 Bits | | 30 |
| ▨ | | | Capabilities Pointer | 34 |
| Bridge Control | | Interrupt Pin | Interrupt Line | 3C |

Figure 17-58. PCI Express PCI-Compatible Configuration Header—Type 1

Section 17.3.8.1, "Common PCI Compatible Configuration Header Registers," describes the registers in the first 16 bytes of the header. This section describes the registers that are unique to the type 1 header beginning at offset 0x10.

### 17.3.8.3.1 PCI Express Base Address Register 0—0x10

Base address register 0 at offset 0x10 is a special fixed 1-Mbyte window that is used for inbound configuration accesses. This window is called the PCI Express configuration and status register base address register (PEXCSRBAR). Note that PEXCSRBAR cannot be updated through the inbound ATMU registers. The PEXCSRBAR is shown in Figure 17-47.

Offset 0x10 Access: Mixed

| | 31 20 | 19 4 | 3 | 2 1 | 0 |
|---|---|---|---|---|---|
| R | ADDRESS | — | PREF | TYPE | MemSp |
| W | | | | | |
| Reset | All zeros | | | | |

**Figure 17-59. PCI Express Base Address Register 0 (PEXCSRBAR)**

Table 17-46 describes the PCI Express configuration and status register base address register.

**Table 17-57. PEXCSRBAR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31–20 | ADDRESS | Indicates the base address that the inbound configuration window occupies. This window is fixed at 1 Mbyte. |
| 19–4 | — | Reserved |
| 3 | PREF | Prefetchable |
| 2–1 | TYPE | Type.<br>00 Locate anywhere in 32-bit address space. |
| 0 | MemSp | Memory space indicator |

### 17.3.8.3.2 PCI Express Primary Bus Number Register—Offset 0x18

The primary bus number register is shown in Figure 17-60.

Offset 0x18 Access: Read/Write

| | 7 0 |
|---|---|
| R | Primary Bus Number |
| W | |
| Reset | All zeros |

**Figure 17-60. PCI Express Primary Bus Number Register**

Table 17-58 describes the primary bus number register fields.

**Table 17-58. PCI Express Primary Bus Number Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 7–0 | Primary Bus Number | Bus that is connected to the upstream interface. Note that this register is programmed during system enumeration; in RC mode this register should remain 0x00. |

#### 17.3.8.3.3 PCI Express Secondary Bus Number Register—Offset 0x19

The secondary bus number register is shown in Figure 17-61.

Offset 0x19 Access: Read/Write

| | 7 … 0 |
|---|---|
| R<br>W | Secondary Bus Number |
| Reset | All zeros |

Figure 17-61. PCI Express Secondary Bus Number Register

Table 17-59 describes the secondary bus number register fields.

Table 17-59. PCI Express Secondary Bus Number Register Field Description

| Bits | Name | Description |
|---|---|---|
| 7–0 | Secondary Bus Number | Bus that is directly connected to the downstream interface. Note that this register is programmed during system enumeration; in RC mode, this register is typically programmed to 0x01. |

#### 17.3.8.3.4 PCI Express Subordinate Bus Number Register—Offset 0x1A

The subordinate bus number register is shown in Figure 17-62.

Offset 0x1A Access: Read/Write

| | 7 … 0 |
|---|---|
| R<br>W | Subordinate Bus Number |
| Reset | All zeros |

Figure 17-62. PCI Express Subordinate Bus Number Register

Table 17-60 describes the subordinate bus number register fields.

Table 17-60. PCI Express Subordinate Bus Number Register Field Description

| Bits | Name | Description |
|---|---|---|
| 7–0 | Subordinate Bus Number | Highest bus number that is on the downstream interface. |

#### 17.3.8.3.5 PCI Express Secondary Latency Timer Register—0x1B

The secondary latency timer register does not apply to PCI Express. It must be read-only and return all zeros when read.

#### 17.3.8.3.6 PCI Express I/O Base Register—0x1C

Note that this device does not support inbound I/O transactions. The I/O base register is shown in Figure 17-62.

Offset 0x1C Access: Read only

| | 7 4 | 3 0 |
|---|---|---|
| R | I/O Start Address | Address Decode Type |
| W | | |
| Reset | All zeros | |

**Figure 17-63. PCI Express I/O Base Register**

Table 17-60 describes the I/O base register fields.

**Table 17-61. PCI Express I/O Base Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 7–4 | I/O Start Address | Specifies bits 15:12 of the I/O space start address |
| 3–0 | Address Decode Type | Specifies the number of I/O address bits.<br>0x00 16-bit I/O address decode<br>0x01 32-bit I/O address decode<br>All other settings reserved. |

#### 17.3.8.3.7 PCI Express I/O Limit Register—0x1D

Note that this device does not support inbound I/O transactions. The I/O limit register is shown in Figure 17-62.

Offset 0x1D Access: Read only

| | 7 4 | 3 0 |
|---|---|---|
| R | I/O Limit Address | Address Decode Type |
| W | | |
| Reset | All zeros | |

**Figure 17-64. PCI Express I/O Limit Register**

Table 17-60 describes the I/O limit register fields.

**Table 17-62. PCI Express I/O Limit Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 7–4 | I/O Limit Address | Specifies bits 15:12 of the I/O space ending address |
| 3–0 | Address Decode Type | Specifies the number of I/O address bits.<br>0x00 16-bit I/O address decode<br>0x01 32-bit I/O address decode<br>All other settings reserved. |

### 17.3.8.3.8 PCI Express Secondary Status Register—0x1E

The PCI Express secondary status register is shown in Figure 17-65. Note that the errors in this register may be masked by corresponding bits in the secondary status interrupt mask register (PEX_SS_INTR_MASK) and that by default all of the errors are masked. See Section 17.3.10.20, “Secondary Status Interrupt Mask Register (RC-Mode Only)—0x5A0,” for more information.

Offset 0x1E Access: Mixed

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| R | DPE | SSE | RMA | RTA | STA | — | | MDPE | — | |
| W | w1c | w1c | w1c | w1c | w1c | | | w1c | | |
| Reset | All zeros | | | | | | | | | |

**Figure 17-65. PCI Express Secondary Status Register**

Table 17-63 describes the PCI Express secondary status register fields.

**Table 17-63. PCI Express Secondary Status Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 15 | DPE | Detected parity error. This bit is set whenever the secondary side receives a poisoned TLP regardless of the state of the parity error response bit. |
| 14 | SSE | Signaled system error. This bit is set when a device sends a ERR_FATAL or ERR_NONFATAL message, provided the SERR enable bit in the command register is set to enable reporting. |
| 13 | RMA | Received master abort. This bit is set when the secondary side receives an unsupported request (UR) completion. |
| 12 | RTA | Received target abort. This bit is set when the secondary side receives a completer abort (CA) completion. |
| 11 | STA | Signaled target abort. This bit is set when the secondary side issues a CA completion. |
| 10–9 | — | Reserved |
| 8 | MDPE | Master data parity error. This bit is set when the parity error response bit is set and the secondary side requestor receives a poisoned completion or poisons a write request. If the parity error response bit is cleared, this bit is never set. |
| 7–0 | — | Reserved |

### 17.3.8.3.9 PCI Express Memory Base Register—0x20

The memory base register is shown in Figure 17-66.

Offset 0x20 Access: Read/Write

| | 15–4 | 3–0 |
|---|---|---|
| R<br>W | Memory Base | — |
| Reset | All zeros | |

**Figure 17-66. PCI Express Memory Base Register**

Table 17-64 describes the memory base register fields.

**Table 17-64. PCI Express Memory Base Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 15–4 | Memory Base | Specifies bits 31:20 of the non-prefetchable memory space start address. Typically used for specifying memory-mapped I/O space.<br>**Note:** Inbound posted transactions hitting into the mem base/limit range are ignored; inbound non-posted transactions hitting into the mem base/limit range results in an unsupported request response. |
| 3–0 | — | Reserved |

### 17.3.8.3.10 PCI Express Memory Limit Register—0x22

The memory limit register is shown in Figure 17-67.

Offset 0x22 Access: Read/Write

| | 15–4 | 3–0 |
|---|---|---|
| R<br>W | Memory Limit | — |
| Reset | All zeros | |

**Figure 17-67. PCI Express Memory Limit Register**

Table 17-65 describes the memory base register fields.

**Table 17-65. PCI Express Memory Limit Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 15–4 | Memory Limit | Specifies bits 31:20 of the non-prefetchable memory space ending address. Typically used for specifying memory-mapped I/O space.<br>**Note:** Inbound posted transactions hitting into the mem base/limit range are ignored; inbound non-posted transactions hitting into the mem base/limit range results in unsupported request response. |
| 3–0 | — | Reserved |

### 17.3.8.3.11 PCI Express Prefetchable Memory Base Register—0x24

The prefetchable memory base register is shown in Figure 17-68.

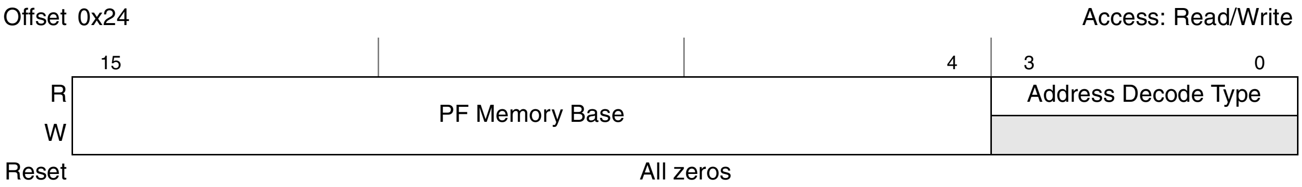


**Figure 17-68. PCI Express Prefetchable Memory Base Register**

Table 17-66 describes the prefetchable memory base register fields.

**Table 17-66. PCI Express Prefetchable Memory Base Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 15–4 | PF Memory Base | Specifies bits 31:20 of the prefetchable memory space start address. |
| 3–0 | Address Decode Type | Specifies the number of prefetchable memory address bits.<br>0x00 32-bit memory address decode<br>0x01 64-bit memory address decode<br>All other settings reserved. |

### 17.3.8.3.12 PCI Express Prefetchable Memory Limit Register—0x26

The prefetchable memory limit register is shown in Figure 17-69.

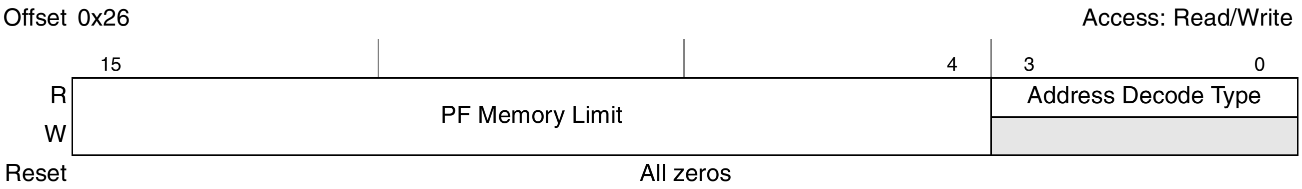


**Figure 17-69. PCI Express Prefetchable Memory Limit Register**

Table 17-67 describes the prefetchable memory limit register fields.

**Table 17-67. PCI Express Prefetchable Memory Limit Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 15–4 | PF Memory Limit | Specifies bits 31:20 of the prefetchable memory space ending address. |
| 3–0 | Address Decode Type | Specifies the number of prefetchable memory address bits.<br>0x00 32-bit memory address decode<br>0x01 64-bit memory address decode<br>All other settings reserved. |

#### 17.3.8.3.13 PCI Express Prefetchable Base Upper 32 Bits Register—0x28

The PCI Express prefetchable memory base upper 32 bits register is shown in Figure 17-70.

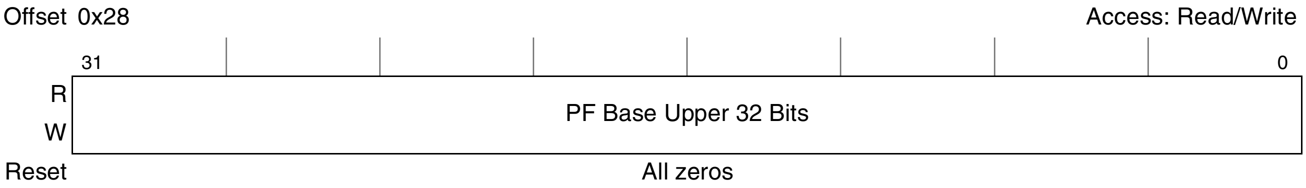


**Figure 17-70. PCI Express Prefetchable Base Upper 32 Bits Register**

Table 17-68 describes the PCI Express prefetchable memory base upper 32 bits register fields.

**Table 17-68. PCI Express Prefetchable Base Upper 32 Bits Register**

| Bits | Name | Description |
|---|---|---|
| 31–0 | PF Base Upper 32 Bits | Specifies bits 64:32 of the prefetchable memory space start address when the address decode type field in the prefetchable memory base register is 0x01. |

#### 17.3.8.3.14 PCI Express Prefetchable Limit Upper 32 Bits Register—0x2C

The PCI Express prefetchable memory base upper 32 bits register is shown in Figure 17-71.

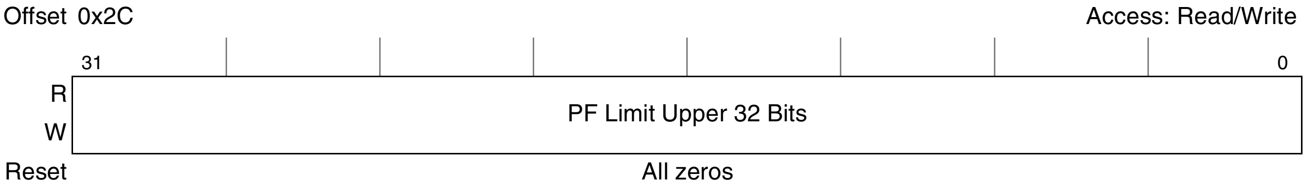


**Figure 17-71. PCI Express Prefetchable Limit Upper 32 Bits Register**

Table 17-69 describes the PCI Express prefetchable memory limit upper 32 bits register fields.

**Table 17-69. PCI Express Prefetchable Limit Upper 32 Bits Register**

| Bits | Name | Description |
|---|---|---|
| 31–0 | PF Limit Upper 32 Bits | Specifies bits 64–32 of the prefetchable memory space ending address when the address decode type field in the prefetchable memory limit register is 0x01. |

#### 17.3.8.3.15 PCI Express I/O Base Upper 16 Bits Register—0x30

Note that this device does not support inbound I/O transactions. The I/O base upper 16 bits register is shown in Figure 17-72.

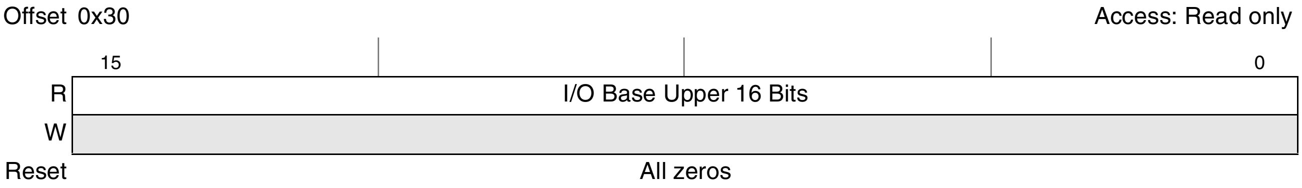


**Figure 17-72. PCI Express I/O Base Upper 16 Bits Register**

Table 17-70 describes the I/O base upper 16 bits register fields.

**Table 17-70. PCI Express I/O Base Upper 16 Bits Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 15–0 | I/O Base Upper 16 Bits | Specifies bits 31–16 of the I/O space start address when the address decode type field in the I/O base register is 0x01. |

### 17.3.8.3.16 PCI Express I/O Limit Upper 16 Bits Register—0x32

Note that this device does not support inbound I/O transactions. The I/O limit upper 16 bits register is shown in Figure 17-73.

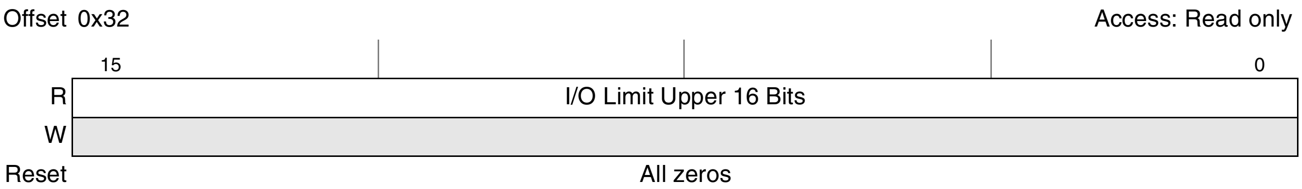


**Figure 17-73. PCI Express I/O Limit Upper 16 Bits Register**

Table 17-71 describes the I/O limit upper 16 bits register fields.

**Table 17-71. PCI Express I/O Limit Upper 16 Bits Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 15–0 | I/O Limit Upper 16 Bits | Specifies bits 31–16 of the I/O space ending address when the address decode type field in the I/O limit register is 0x01. |

### 17.3.8.3.17 Capabilities Pointer Register—0x34

The capabilities pointer identifies additional functionality supported by the device.

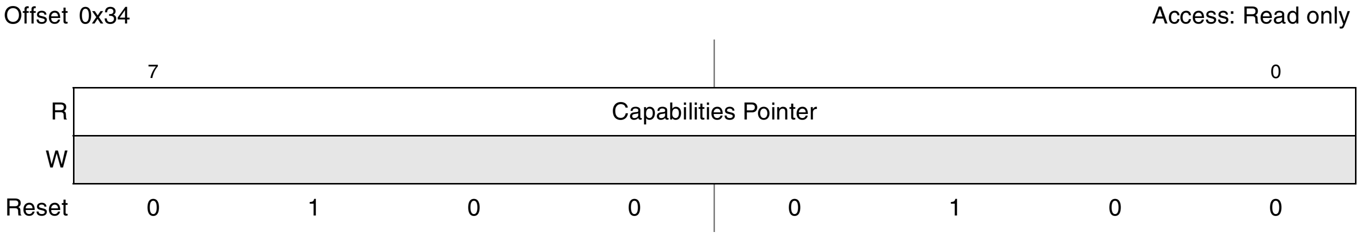


**Figure 17-74. Capabilities Pointer Register**

**Table 17-72. Capabilities Pointer Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 7–0 | Capabilities Pointer | The capabilities pointer provides the offset (0x44) for additional PCI-compatible registers above the common 64-byte header. Refer to Section 17.3.9, “PCI Compatible Device-Specific Configuration Space,” for more information. |

### 17.3.8.3.18 PCI Express Interrupt Line Register—0x3C

The interrupt line register is used by device drivers and OS software to communicate interrupt line routing information. Values in this register are programmed by system software and are system specific.

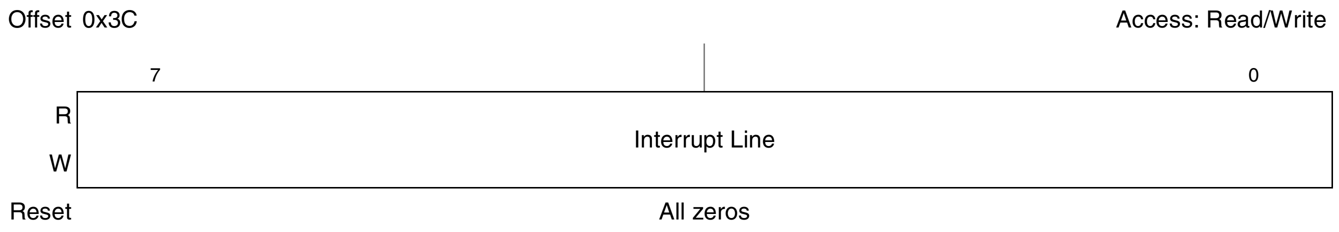


**Figure 17-75. PCI Express Interrupt Line Register**

**Table 17-73. PCI Express Interrupt Line Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 7–0 | Interrupt Line | Used to communicate interrupt line routing information. |

### 17.3.8.3.19 PCI Express Interrupt Pin Register—0x3D

The interrupt pin register identifies the legacy interrupt (INTx) messages the device (or function) uses.

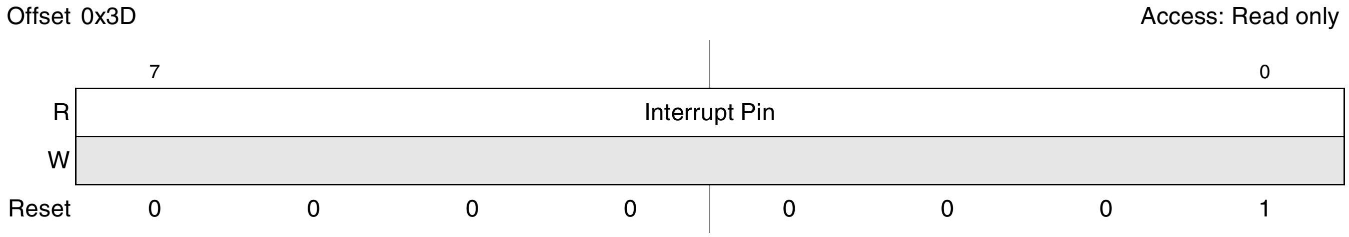


**Figure 17-76. PCI Express Interrupt Pin Register**

**Table 17-74. PCI Express Interrupt Pin Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 7–0 | Interrupt pin | Legacy INTx message used by this device.<br>0x00 This device does not use legacy interrupt (INTx) messages.<br>0x01 INTA<br>0x02 INTB<br>0x03 INTC<br>0x04 INTD<br>all others Reserved. |

### 17.3.8.3.20 PCI Express Bridge Control Register—0x3E

The PCI Express bridge control register is shown in Figure 17-77.

Offset 0x3E | Access: Read/Write

| | 15–7 | 6 | 5–4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| R<br>W | — | Scnd_RST | — | VGA_EN | ISA_EN | SERR_EN | PER |
| Reset | All zeros | | | | | | |

**Figure 17-77. PCI Express Bridge Control Register**

Table 17-75 describes the PCI Express bridge control register fields.

**Table 17-75. PCI Express Bridge Control Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 15–7 | — | Reserved |
| 6 | Scnd_RST | Secondary bus reset |
| 5–4 | — | Reserved |
| 3 | VGA_EN | VGA enable |
| 2 | ISA_EN | ISA enable |
| 1 | SERR_EN | SERR enable. This bit controls the propagation of ERR_COR, ERR_NONFATAL, and ERR_FATAL responses received on the secondary side. |
| 0 | PER | Parity error response. |

### 17.3.9 PCI Compatible Device-Specific Configuration Space

The PCI compatible device-specific configuration space is a PCI compatible configuration space from 0x40 to 0xFF (just above the 64-byte PCI-compatible configuration header).

▨ Reserved

<table>
<tr><th colspan="4"></th><th>Address Offset (Hex)</th></tr>
<tr><td colspan="4">PCI-Compatible Configuration Header<br>(See Section 17.3.8, “PCI Compatible Configuration Headers,” for more information.)</td><td>00<br>3F</td></tr>
<tr><td colspan="4">▨</td><td>40</td></tr>
<tr><td colspan="2">Power Mgmt Capabilities</td><td>Next Pointer (0x4C)</td><td>Power Mgmt Capability ID</td><td>44</td></tr>
<tr><td>Data</td><td>▨</td><td colspan="2">Power Management Status & Control</td><td>48</td></tr>
<tr><td colspan="2">PCI Express Capabilities</td><td>Next Pointer<br>(0x70—EP mode)<br>(NULL—RC mode)</td><td>PCI Express Capability ID</td><td>4C</td></tr>
<tr><td colspan="4">Device Capabilities</td><td>50</td></tr>
<tr><td colspan="2">Device Status</td><td colspan="2">Device Control</td><td>54</td></tr>
<tr><td colspan="4">Link Capabilities</td><td>58</td></tr>
<tr><td colspan="2">Link Status</td><td colspan="2">Link Control</td><td>5C</td></tr>
<tr><td colspan="4">Slot Capabilities</td><td>60</td></tr>
<tr><td colspan="2">Slot Status</td><td colspan="2">Slot Control</td><td>64</td></tr>
<tr><td colspan="2">▨</td><td colspan="2">Root Control (RC mode only)</td><td>68</td></tr>
<tr><td colspan="4">Root Status</td><td>6C</td></tr>
<tr><td colspan="2">MSI Message Control</td><td>Next Pointer (NULL)</td><td>MSI Message Capability ID</td><td>70</td></tr>
<tr><td colspan="4">MSI Message Address</td><td>74</td></tr>
<tr><td colspan="4">MSI Upper Message Address</td><td>78</td></tr>
<tr><td colspan="2">▨</td><td colspan="2">MSI Message Data</td><td>7C</td></tr>
<tr><td colspan="4">▨</td><td>80<br>FF</td></tr>
</table>

**Figure 17-78. PCI Compatible Device-Specific Configuration Space**

### 17.3.9.1 PCI Express Power Management Capability ID Register—0x44

The PCI Express power management capability ID register is shown in Figure 17-79.

Offset 0x44 Access: Read only

| | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|
| R | Power Mgmt Capability ID | | | | | | | |
| W | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

**Figure 17-79. PCI Express Power Management Capability ID Register**

**Table 17-76. PCI Express Power Management Capability ID Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 7–0 | Power Mgmt Capability ID | Power Management = 0x01 |

### 17.3.9.2 PCI Express Power Management Capabilities Register—0x46

The PCI Express power management capabilities register is shown in Figure 17-80.

Offset 0x46 Access: Read only

| | 15 | | | | 11 | 10 | 9 | 8 | | 6 | 5 | 4 | 3 | 2 | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | PME Support | | | | | D2 | D1 | AUX Curr | | | DSI | — | PME CLK | Version | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

**Figure 17-80. PCI Express Power Management Capabilities Register**

**Table 17-77. PCI Express Power Management Capabilities Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 15–11 | PME Support | Indicates the power states that this device supports |
| 10 | D2 | D2 Support |
| 9 | D1 | D1 Support |
| 8–6 | AUX Curr | AUX Current |
| 5 | DSI | Device Specific Initialization |
| 4 | — | Reserved |
| 3 | PME CLK | Does not apply to PCI Express. |
| 2–0 | Version | Set to 0x2 for this version of the specification. |

### 17.3.9.3 PCI Express Power Management Status and Control Register—0x48

The PCI Express power management status and control register is shown in Figure 17-81.

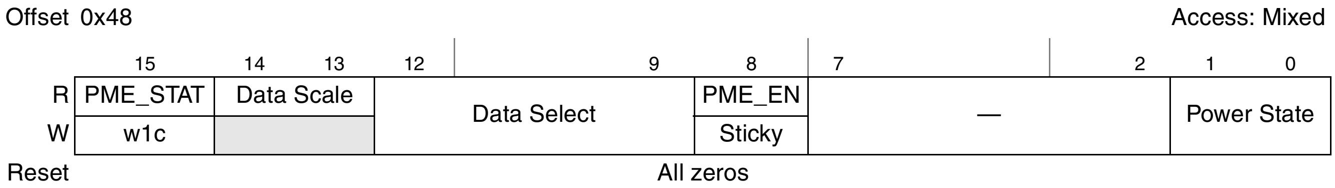


**Figure 17-81. PCI Express Power Management Status and Control Register**

**Table 17-78. PCI Express Status and Control Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 15 | PME_STAT | PME Status |
| 14–13 | Data Scale | Obtained directly from *PCI Express™ Base Specification, Revision 1.0a* |
| 12–9 | Data Select | Obtained directly from *PCI Express™ Base Specification, Revision 1.0a* |
| 8 | PME_EN | PME Enable |
| 7–2 | — | Reserved |
| 1–0 | Power State | Power state. Indicates the current power state of the function.<br>0x00 D0<br>0x01 D1<br>0x02 D2<br>0x03 D3 |

### 17.3.9.4 PCI Express Power Management Data Register—0x4B

The PCI Express power management data register is shown in Figure 17-82.

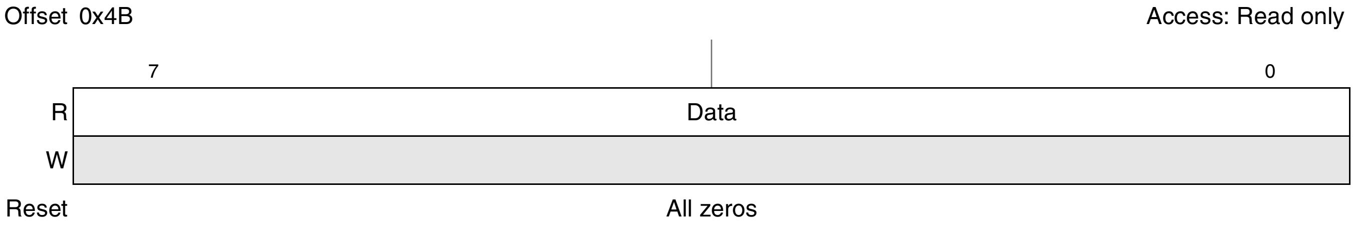


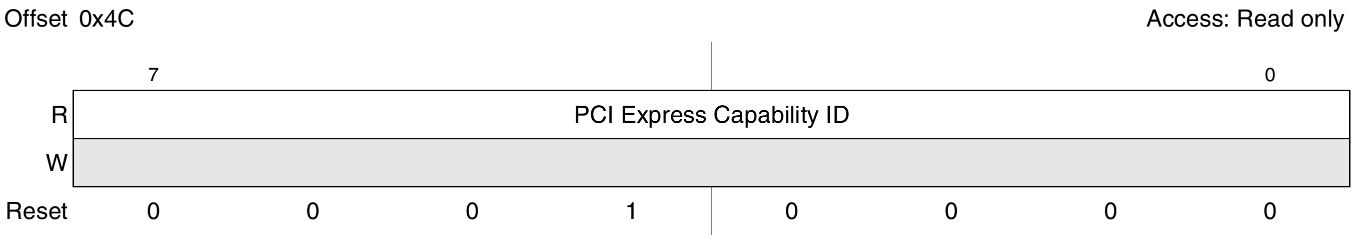
**Figure 17-82. PCI Express Power Management Data Register**

**Table 17-79. PCI Express Power Management Data Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 7–0 | Data | Obtained from *PCI Express™ Base Specification, Revision 1.0a* |

### 17.3.9.5 PCI Express Capability ID Register—0x4C

The PCI Express capability ID register is shown in Figure 17-83.



Offset 0x4C | Access: Read only

| | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|
| R | PCI Express Capability ID | | | | | | | |
| W | | | | | | | | |
| Reset | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

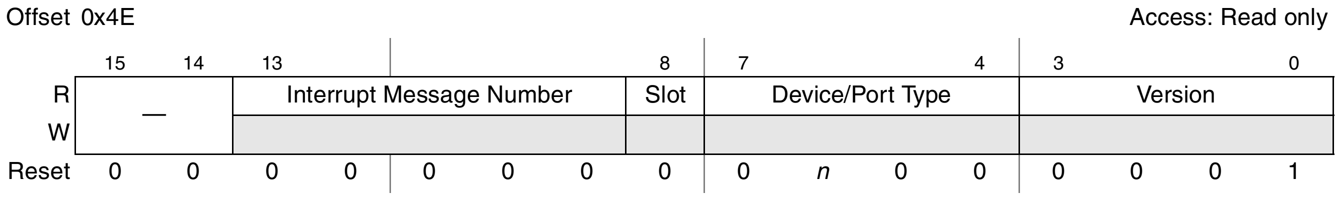
**Figure 17-83. PCI Express Capability ID Register**

**Table 17-80. PCI Express Capability ID Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 7–0 | PCI Express Capability ID | PCI Express = 0x10 |

### 17.3.9.6 PCI Express Capabilities Register—0x4E

The PCI Express capabilities register is shown in Figure 17-84.



Offset 0x4E | Access: Read only

| | 15 | 14 | 13 | | | | | 8 | 7 | | | 4 | 3 | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | — | | Interrupt Message Number | | | | | Slot | Device/Port Type | | | | Version | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | *n* | 0 | 0 | 0 | 0 | 0 | 1 |

**Figure 17-84. PCI Express Capabilities Register**

**Table 17-81. PCI Express Capabilities Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 15–14 | — | Reserved |
| 13–9 | Interrupt Message Number | If this function is allocated more than one MSI interrupt number, then this register is required to contain the offset between the base Message Data and the MSI Message that is generated when any of the status bits in either the Slot Status register or the Root Port Status register, of this capability structure, are set. |
| 8 | Slot | Slot Implemented (RC mode only) |
| 7–4 | Device/Port Type | 0100 (RC mode)<br>0000 (EP mode) |
| 3–0 | Capability Version | Indicates the defined PCI Express capability structure version number. Must be 1h for 1.0, 1.0a, and 1.1 specification. |

### 17.3.9.7 PCI Express Device Capabilities Register—0x50

The PCI Express device capabilities register is shown in Figure 17-85.

Offset 0x50 Access: Read only

| | 31 | | | 28 | 27 | 26 | 25 | | | | | | | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | — | | | | CSPLS | | CSPLV | | | | | | | | — | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |

| | 15 | 14 | 13 | 12 | 11 | | 9 | 8 | | 6 | 5 | 4 | 3 | 2 | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | — | PIP | AIP | ABP | EP_L1_LAT | | | EP_L0s_LAT | | | ET | PHAN_FCT | | MAX_PL_SIZE_SUP | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

**Figure 17-85. PCI Express Device Capabilities Register**

**Table 17-82. PCI Express Device Capabilities Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 31–28 | — | Reserved |
| 27–26 | CSPLS | Captured Slot Power Limit Scale |
| 25–18 | CSPLV | Captured Slot Power Limit Value |
| 17–15 | — | Reserved |
| 14 | PIP | Power Indicator Present |
| 13 | AIP | Attention Indicator Present |
| 12 | ABP | Attention Button Present |
| 11–9 | EP_L1_LAT | Endpoint L1 Acceptable Latency |
| 8–6 | EP_L0s_LAT | Endpoint L0s Acceptable Latency |
| 5 | ET | Extended Tag Field Supported |
| 4–3 | PHAN_FCT | Phantom Functions Supported |
| 2–0 | MAX_PL_SIZE_SUP | Maximum payload size supported. 001 = 256-bytes |

### 17.3.9.8 PCI Express Device Control Register—0x54

The PCI Express device control register is shown in Figure 17-86.

Offset 0x54 Access: Read/write

| | 15 | 14 | | 12 | 11 | 10 | 9 | 8 | 7 | | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | MAX_READ_SIZE | | | NSE | APE | PFE | ETE | MAX_PAYLOAD_SIZE | | | RO | URR | FER | NFE<br>R | CER |
| Reset | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

**Figure 17-86. PCI Express Device Control Register**

Table 17-83. PCI Express Device Control Register Field Description

| Bits | Name | Description |
|---|---|---|
| 15 | — | Reserved |
| 14–12 | MAX_READ_SIZE | Maximum read request size |
| 11 | NSE | No snoop enable |
| 10 | APE | AUX power PM enable |
| 9 | PFE | Phantom functions enable |
| 8 | ETE | Extended tag field enable |
| 7–5 | MAX_PAYLOAD_SIZE | Maximum payload size |
| 4 | RO | Relaxed ordering |
| 3 | URR | Unsupported request reporting |
| 2 | FER | Fatal error reporting |
| 1 | NFER | Non-fatal error reporting |
| 0 | CER | Correctable error reporting |

### 17.3.9.9 PCI Express Device Status Register—0x56

The PCI Express device status register is shown in Figure 17-87.

Offset 0x56 Access: Mixed

| | 15 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| R | — | TP | APD | URD | FED | NFED | CED |
| W | | | | w1c | w1c | w1c | w1c |
| Reset | All zeros | | | | | | |

Figure 17-87. PCI Express Device Status Register

Table 17-84. PCI Express Device Status Register Field Description

| Bits | Name | Description |
|---|---|---|
| 15–6 | — | Reserved |
| 5 | TP | Transactions pending |
| 4 | APD | AUX power detected |
| 3 | URD | Unsupported request detected |
| 2 | FED | Fatal error detected |
| 1 | NFED | Non-fatal error detected |
| 0 | CED | Correctable error detected |

### 17.3.9.10 PCI Express Link Capabilities Register—0x58

The PCI Express link capabilities register is shown in Figure 17-88.

Offset 0x58 Access: Read only

| | 31–24 | 23–18 | 17–15 | 14–12 | 11–10 | 9–4 | 3–0 |
|---|---|---|---|---|---|---|---|
| R | Port Number | — | L1_EX_LA | L0s_EX_LAT | ASPM | MAX_LINK_W | MAX_LINK_SP |
| W | | | | | | | |
| Reset | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 | 1 1 1 | 0 1 1 | 0 1 | 0 0 1 0 0 0 | 0 0 0 1 |

**Figure 17-88. PCI Express Link Capabilities Register**

**Table 17-85. PCI Express Link Capabilities Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 31–24 | Port Number | — |
| 23–18 | — | Reserved |
| 17–15 | L1_EX_LAT | L1 exit latency |
| 14–12 | L0s_EX_LAT | L0s exit latency |
| 11–10 | ASPM | Active state power management (ASPM) Support |
| 9–4 | MAX_LINK_W | Maximum link width |
| 3–0 | MAX_LINK_SP | Maximum link speed<br>0001 2.5 GT/s link |

### 17.3.9.11 PCI Express Link Control Register—0x5C

The PCI Express link control register is shown in Figure 17-89.

Offset 0x5C Access: Mixed

| | 15–8 | 7 | 6 | 5 | 4 | 3 | 2 | 1–0 |
|---|---|---|---|---|---|---|---|---|
| R | — | EXT_SYNC | CCC | | LD | RCB | — | ASPM_CTL |
| W | | | | RL | | | | |
| Reset | All zeros | | | | | | | |

**Figure 17-89. PCI Express Link Control Register**

**Table 17-86. PCI Express Link Control Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 15–8 | — | Reserved |
| 7 | EXT_SYNC | Extended synch |
| 6 | CCC | Common clock configuration |
| 5 | RL | Retrain link (Reserved for EP devices). In RC mode, setting this bit initiates link retraining by directing the Physical Layer LTSSM to the Recovery state; reads of this bit always return 0. |

Table 17-86. PCI Express Link Control Register Field Description (continued)

| Bits | Name | Description |
|---|---|---|
| 4 | LD | Link disable (Reserved for EP devices) |
| 3 | RCB | Read completion boundary |
| 2 | — | Reserved |
| 1–0 | ASPM_CTL | Active state power management (ASPM) control |

### 17.3.9.12 PCI Express Link Status Register—0x5E

The PCI Express link status register is shown in Figure 17-90.

Offset 0x5E

Access: Read only

| | 15 | | 13 | 12 | 11 | 10 | 9 | | | | | 4 | 3 | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | — | | | SCC | LT | — | NEG_LINK_W | | | | | | LINK_SP | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |

Figure 17-90. PCI Express Link Status Register

Table 17-87. PCI Express Link Status Register Field Description

| Bits | Name | Description |
|---|---|---|
| 15–13 | — | Reserved |
| 12 | SCC | Slot clock configuration |
| 11 | LT | Link training |
| 10 | — | Reserved. |
| 9–4 | NEG_LINK_W | Negotiated link width |
| 3–0 | LINK_SP | Link speed. |

### 17.3.9.13 PCI Express Slot Capabilities Register—0x60

The PCI Express slot capabilities register is shown in Figure 17-91.

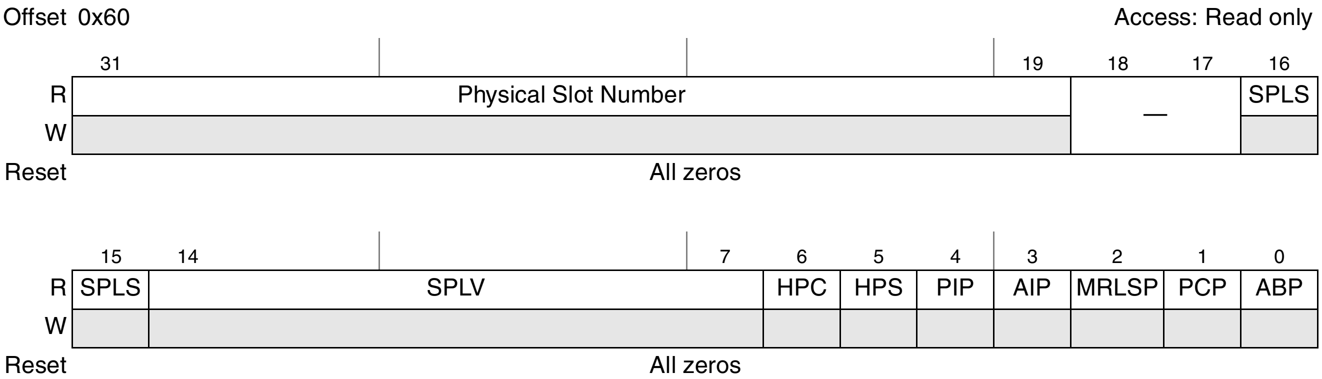


**Figure 17-91. PCI Express Slot Capabilities Register**

**Table 17-88. PCI Express Slot Capabilities Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 31–19 | Physical Slot Number | This hardware initialized field indicates the physical slot number attached to this Port. This field must be hardware initialized to a value that assigns a slot number that is globally unique within the chassis. These registers should be initialized to 0 for Ports connected to devices that are either integrated on the system board or integrated within the same silicon as the Switch device or Root Port. |
| 18–17 | — | Reserved |
| 16–15 | SPLS | Slot power limit scale. |
| 14–7 | SPLV | Slot power limit value. |
| 6 | HPD | Hot plug capable. |
| 5 | HPS | Hot plug surprise. |
| 4 | PIP | Power indicator present. |
| 3 | AIP | Attention indicator present. |
| 2 | MRLSP | MRL sensor present. |
| 1 | PCP | Power controller present. |
| 0 | ABP | Attention button present. |

### 17.3.9.14 PCI Express Slot Control Register—0x64

The PCI Express slot control register is shown in Figure 17-92.

Offset 0x64 Access: Read/Write

| | 15–11 | 10 | 9–8 | 7–6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | PCC | PIC | AIC | HPIE | CCIE | PDCE | MRLSCE | PFDE | ABPE |
| Reset | All zeros | | | | | | | | | |

**Figure 17-92. PCI Express Slot Control Register**

**Table 17-89. PCI Express Slot Control Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 15–11 | — | Reserved |
| 10 | PCC | Power controller control. |
| 9–8 | PIC | Power indicator control. |
| 7–6 | AIC | Attention indicator control. |
| 5 | HPIE | Hot plug interrupt enable. |
| 4 | CCIE | Command completed interrupt enable. |
| 3 | PDCE | Presence detect changed enable. |
| 2 | MRLSCE | MRL sensor changed enable. |
| 1 | PFDE | Power fault detected enable. |
| 0 | ABPE | Attention button pressed enable. |

### 17.3.9.15 PCI Express Slot Status Register—0x66

The PCI Express slot status register is shown in Figure 17-93.

Offset 0x66 Access: Mixed

| | 15–7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| R | — | PDS | MRLSS | CC | PDC | MRLSC | PFD | ABP |
| W | | | | w1c | w1c | w1c | w1c | w1c |
| Reset | 0 0 0 0 0 0 0 0 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

**Figure 17-93. PCI Express Slot Status Register**

**Table 17-90. PCI Express Slot Status Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 15–7 | — | Reserved |
| 6 | PDS | Presence detect state. This bit indicates the presence of a card in the slot.<br>0 Slot empty<br>1 Card is present |

**Table 17-90. PCI Express Slot Status Register Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 5 | MRLSS | MRL sensor state.<br>0 MRL closed<br>1 MRL open |
| 4 | CC | Command completed. |
| 3 | PDC | Presence detect changed. |
| 2 | MRLSC | MRL sensor changed. |
| 1 | PFD | Power fault detected. |
| 0 | ABP | Attention button pressed. |

## 17.3.9.16 PCI Express Root Control Register (RC Mode Only)—0x68

The PCI Express root control register is shown in Figure 17-94.

Offset 0x68 Access: Read/Write

| | 15–4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|
| R<br>W | — | PMEIE | SEFEE | SENFEE | SECEE |
| Reset | All zeros | | | | |

**Figure 17-94. PCI Express Root Control Register**

**Table 17-91. PCI Express Root Control Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 15–4 | — | Reserved |
| 3 | PMEIE | PME interrupt enable. |
| 2 | SEFEE | System error on fatal error enable. |
| 1 | SENFEE | System error on non-fatal error enable. |
| 0 | SECEE | System error on correctable error enable. |

## 17.3.9.17 PCI Express Root Status Register (RC Mode Only)—0x6C

The PCI Express root status register is shown in Figure 17-95.

Offset 0x6C　　　　　　　　　　　　　　　　　　　　　　　　　　　　Access: Mixed

| | 31–18 | 17 | 16 |
|---|---|---|---|
| R | — | PMEP | PMES |
| W | | | w1c |
| Reset | All zeros | | |

| | 15–0 |
|---|---|
| R | PME Requestor ID |
| W | |
| Reset | All zeros |

**Figure 17-95. PCI Express Root Status Register**

**Table 17-92. PCI Express Root Status Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 31–18 | — | Reserved |
| 17 | PMEP | PME pending. |
| 16 | PMES | PME status. |
| 15–0 | PME Requestor ID | PME requestor ID. |

### 17.3.9.18 PCI Express MSI Message Capability ID Register (EP Mode Only)—0x70

The PCI Express MSI message capability ID register is shown in Figure 17-96.

Offset 0x70　　　　　　　　　　　　　　　　　　　　　　　　　　　　Access: Read only

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| R | MSI Message Capability ID | | | | | | | |
| W | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |

**Figure 17-96. PCI Express Capability ID Register**

**Table 17-93. PCI Express Capability ID Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 7–0 | MSI Message Capability ID | MSI Message = 0x05 |

### 17.3.9.19 PCI Express MSI Message Control Register (EP Mode Only)—0x72

The PCI Express MSI message control register is shown in Figure 17-97.

Offset 0x72 Access: Mixed

| | 15 | | | | | | | 8 | 7 | 6 | | 4 | 3 | | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | — | | | | | | | | 64AC | MME | | | MMC | | | MSIE |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |

**Figure 17-97. PCI Express MSI Message Control Register**

**Table 17-94. PCI Express MSI Message Control Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 15–8 | — | Reserved |
| 7 | 64AC | 64-bit address capable. |
| 6–4 | MME | Multiple message enable. |
| 3–1 | MMC | Multiple message capable. |
| 0 | MSIE | MSI enable. |

### 17.3.9.20 PCI Express MSI Message Address Register (EP Mode Only)—0x74

The PCI Express MSI message address register is shown in Figure 17-98.

Offset 0x74 Access: Mixed

| | 31 | 2 | 1 | 0 |
|---|---|---|---|---|
| R | Message Address | | 00 | |
| W | | | | |
| Reset | All zeros | | | |

**Figure 17-98. PCI Express MSI Message Address Register**

**Table 17-95. PCI Express MSI Message Address Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 31–2 | Message Address | System-specified message address |
| 1–0 | 00 | Always returns 00 on reads; write operations have no effect. |

### 17.3.9.21 PCI Express MSI Message Upper Address Register (EP Mode Only)—0x78

The PCI Express MSI message upper address register is shown in Figure 17-99.

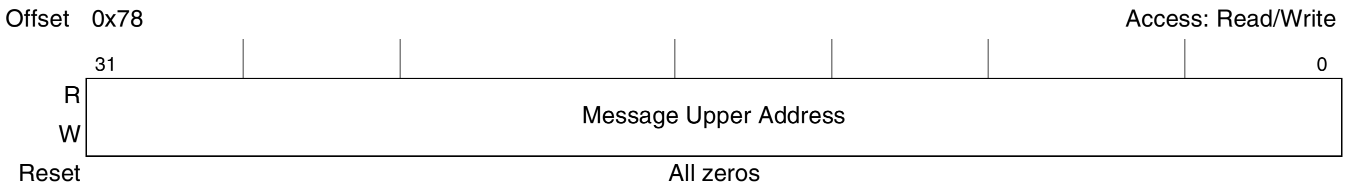


**Figure 17-99. PCI Express MSI Message Upper Address Register**

**Table 17-96. PCI Express MSI Message Upper Address Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 31–0 | Message Upper Address | System-specified message upper address |

### 17.3.9.22 PCI Express MSI Message Data Register (EP Mode Only)—0x7C

The PCI Express MSI message data register is shown in Figure 17-100.

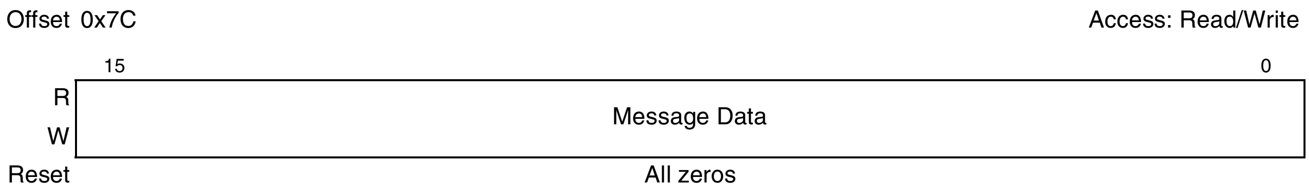


**Figure 17-100. PCI Express MSI Message Data Register**

**Table 17-97. PCI Express MSI Message Data Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 15–0 | Message Data | System-specified message. |

### 17.3.10 PCI Express Extended Configuration Space

▨ Reserved

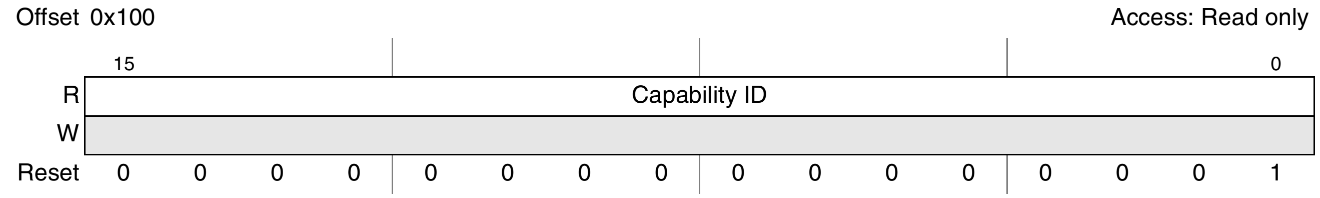
| | | Address Offset (Hex) |
|---|---|---|
| PCI Compatible Configuration Header<br>(See Section 17.3.8, “PCI Compatible Configuration Headers,” for more information.) | | 000<br>03F |
| PCI-Compatible Device-Specific Configuration Space<br>(See Section 17.3.9, “PCI Compatible Device-Specific Configuration Space,” for more information.) | | 040<br>0FF |
| Next Capability Offset (NULL)/Capability Version | Advanced Error Reporting Capability ID | 100 |
| Uncorrectable Error Status | | 104 |
| Uncorrectable Error Mask | | 108 |
| Uncorrectable Error Severity | | 10C |
| Correctable Error Status | | 110 |
| Correctable Error Mask | | 114 |
| Advanced Error Capabilities and Control | | 118 |
| Header Log | | 11C<br>120<br>124<br>128 |
| Root Error Command | | 12C |
| Root Error Status | | 130 |
| Error Source ID | Correctable Error Source ID | 134 |
| ▨ | | 138<br>3FF |
| PCI Express Controller Internal CSRs[1] | | 400<br>6FF |
| ▨ | | 700<br>FFF |

**Figure 17-101. PCI Express Extended Configuration Space**

[1] Note that the PCI Express Controller Internal CSRs are not accessible by inbound PCI Express configuration transactions. Attempts to access these registers returns all 0s.

### 17.3.10.1 PCI Express Advanced Error Reporting Capability ID Register—0x100

The PCI Express advanced error reporting capability ID register is shown in Figure 17-102.



Offset 0x100 Access: Read only

| | 15 | | | | | | | | | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Capability ID | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

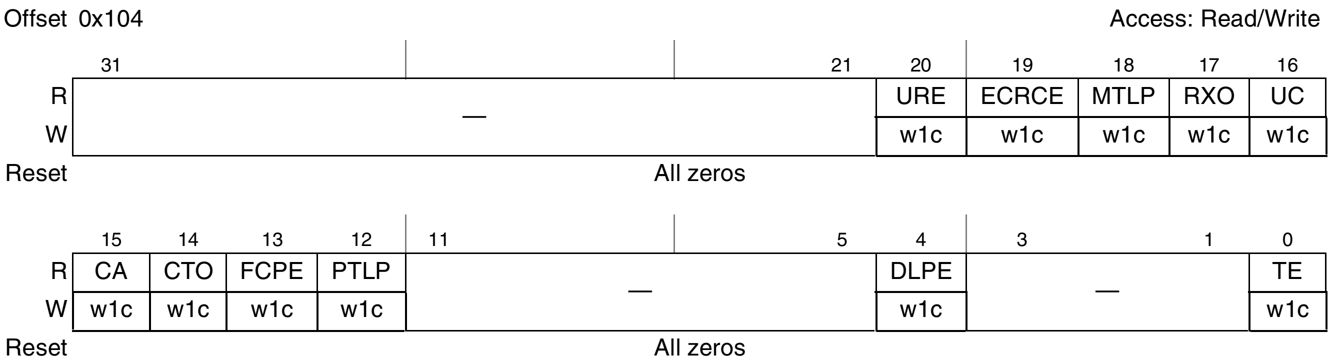
**Figure 17-102. PCI Express Advanced Error Reporting Capability ID Register**

**Table 17-98. PCI Express Advanced Error Reporting Capability ID Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 15–0 | Capability ID | Advanced error reporting capability = 0x0001 |

### 17.3.10.2 PCI Express Uncorrectable Error Status Register—0x104

The PCI Express uncorrectable error status register is shown in Figure 17-103.



Offset 0x104 Access: Read/Write

| | 31–21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|
| R | — | URE | ECRCE | MTLP | RXO | UC |
| W | | w1c | w1c | w1c | w1c | w1c |
| Reset | All zeros | | | | | |

| | 15 | 14 | 13 | 12 | 11–5 | 4 | 3–1 | 0 |
|---|---|---|---|---|---|---|---|---|
| R | CA | CTO | FCPE | PTLP | — | DLPE | — | TE |
| W | w1c | w1c | w1c | w1c | | w1c | | w1c |
| Reset | All zeros | | | | | | | |

**Figure 17-103. PCI Express Uncorrectable Error Status Register**

**Table 17-99. PCI Express Uncorrectable Error Status Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 31–21 | — | Reserved |
| 20 | URE | Unsupported request error status. |
| 19 | ECRCE | ECRC error status. |
| 18 | MTLP | Malformed TLP status. |
| 17 | RXO | Receiver overflow status. |
| 16 | UC | Unexpected completion status. |
| 15 | CA | Completer abort status. |
| 14 | CTO | Completion timeout status. Note that a completion timeout error is a fatal error. If a completion timeout error is detected, the system has become unstable. Hot reset is recommended to restore stability of the system. |

**Table 17-99. PCI Express Uncorrectable Error Status Register Field Description (continued)**

| Bits | Name | Description |
|---|---|---|
| 13 | FCPE | Flow control protocol error status. |
| 12 | PTLP | Poisoned TLP status. |
| 11–5 | — | Reserved |
| 4 | DLPE | Data link protocol error status. |
| 3–1 | — | Reserved |
| 0 | TE | Training error status. |

### 17.3.10.3 PCI Express Uncorrectable Error Mask Register—0x108

The PCI Express uncorrectable error mask register is shown in Figure 17-104.

Offset 0x108 Access: Read/Write

| | 31 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| R<br>W | — | | UREM | ECRCEM | MTLPM | RXOM | UCM |
| Reset | All zeros | | | | | | |

| | 15 | 14 | 13 | 12 | 11 | 5 | 4 | 3 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | CAM | CTOM | FCPEM | PTLPM | — | | DLPEM | — | | TEM |
| Reset | All zeros | | | | | | | | | |

**Figure 17-104. PCI Express Uncorrectable Error Mask Register**

**Table 17-100. PCI Express Uncorrectable Error Mask Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 31–21 | — | Reserved |
| 20 | UREM | Unsupported request error mask. |
| 19 | ECRCEM | ECRC error mask. |
| 18 | MTLPM | Malformed TLP mask. |
| 17 | RXOM | Receiver overflow mask. |
| 16 | UCM | Unexpected completion mask. |
| 15 | CAM | Completer abort mask. |
| 14 | CTOM | Completion timeout mask. |
| 13 | FCPEM | Flow control protocol error mask. |
| 12 | PTLPM | Poisoned TLP mask. |
| 11–5 | — | Reserved |
| 4 | DLPEM | Data link protocol error mask. |

**Table 17-100. PCI Express Uncorrectable Error Mask Register Field Description (continued)**

| Bits | Name | Description |
|---|---|---|
| 3–1 | — | Reserved |
| 0 | TEM | Training error mask. |

### 17.3.10.4 PCI Express Uncorrectable Error Severity Register—0x10C

The PCI Express uncorrectable error severity register is shown in Figure 17-105.

Offset 0x10C Access: Read/Write

| | 31 | | | | | | | | | | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | | | | | | | | | | | URES | ECRCES | MTLPS | RXOS | UCS |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |

| | 15 | 14 | 13 | 12 | 11 | | | | | | 5 | 4 | 3 | | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | CAS | CTOS | FCPES | PTLPS | — | | | | | | | DLPES | — | | | TES |
| Reset | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |

**Figure 17-105. PCI Express Uncorrectable Error Severity Register**

**Table 17-101. PCI Express Uncorrectable Error Severity Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 31–21 | — | Reserved |
| 20 | URES | Unsupported request error severity. |
| 19 | ECRCES | ECRC error severity. |
| 18 | MTLPS | Malformed TLP severity. |
| 17 | RXOS | Receiver overflow severity. |
| 16 | UCS | Unexpected completion severity. |
| 15 | CAS | Completer abort severity. |
| 14 | CTOS | Completion timeout severity. |
| 13 | FCPES | Flow control protocol error severity. |
| 12 | PTLPS | Poisoned TLP severity. |
| 11–5 | — | Reserved |
| 4 | DLPES | Data link protocol error severity. |
| 3–1 | — | Reserved |
| 0 | TES | Training error severity. |

### 17.3.10.5 PCI Express Correctable Error Status Register—0x110

The PCI Express correctable error status register is shown in Figure 17-106.

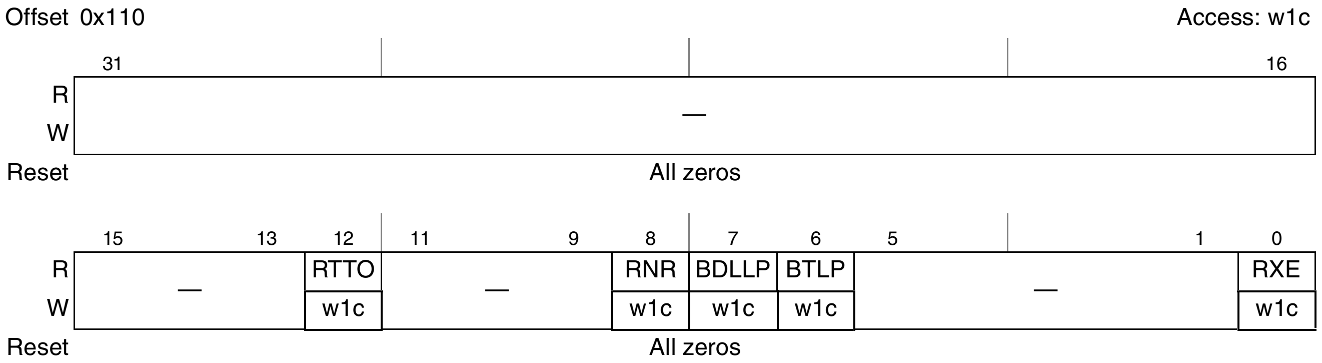


**Figure 17-106. PCI Express Correctable Error Status Register**

**Table 17-102. PCI Express Correctable Error Status Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 31–13 | — | Reserved |
| 12 | RTTO | Replay timer timeout status |
| 11–9 | — | Reserved |
| 8 | RNR | REPLAY_NUM Rollover status |
| 7 | BDLLP | Bad DLLP status |
| 6 | BTLP | Bad TLP status |
| 5–1 | — | Reserved |
| 0 | RXE | Receiver error status |

### 17.3.10.6 PCI Express Correctable Error Mask Register—0x114

The PCI Express correctable error mask register is shown in Figure 17-107.

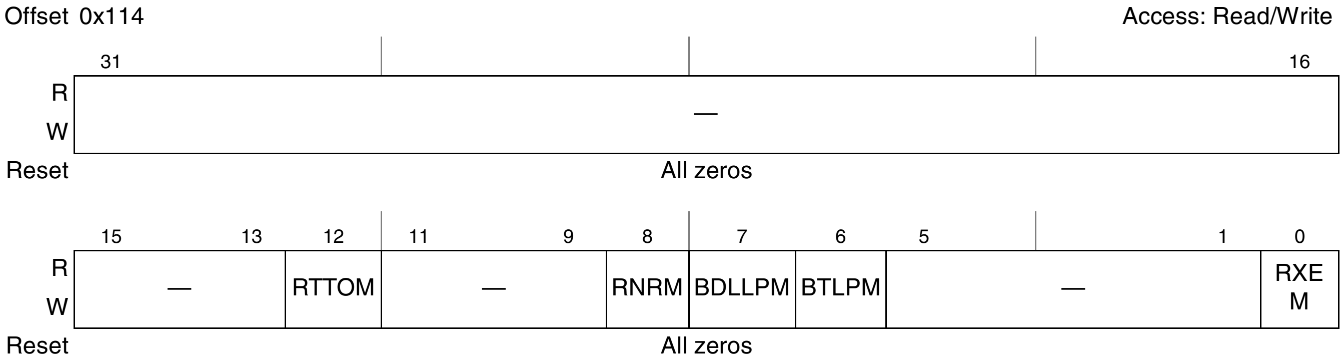


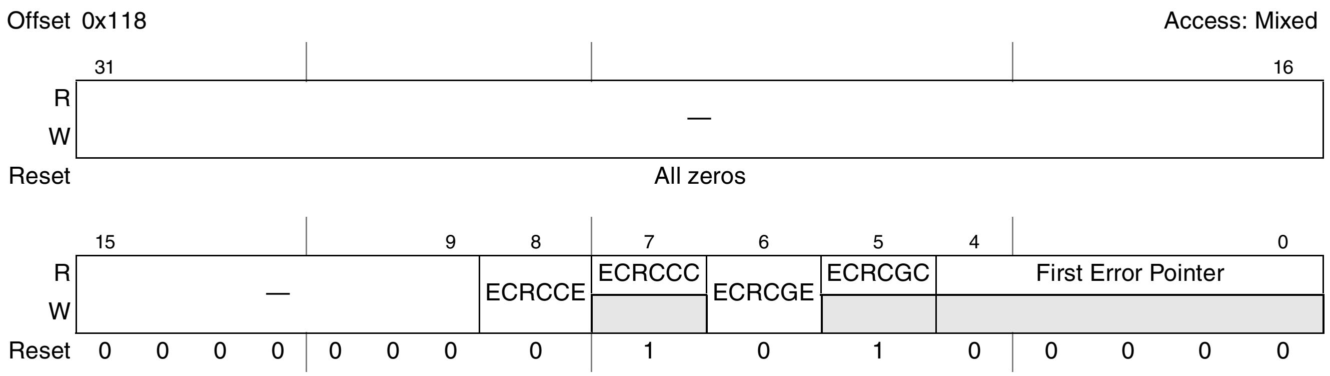
**Figure 17-107. PCI Express Correctable Error Mask Register**

Table 17-103. PCI Express Correctable Error Mask Register Field Description

| Bits | Name | Description |
|---|---|---|
| 31–13 | — | Reserved |
| 12 | RTTOM | Replay timer timeout mask |
| 11–9 | — | Reserved |
| 8 | RNRM | REPLAY_NUM Rollover mask |
| 7 | BDLLPM | Bad DLLP mask |
| 6 | BTLPM | Bad TLP mask |
| 5–1 | — | Reserved |
| 0 | RXEM | Receiver error mask |

### 17.3.10.7 PCI Express Advanced Error Capabilities and Control Register—0x118

The PCI Express advanced error capabilities and control register is shown in Figure 17-108.



Offset 0x118 Access: Mixed

| | 31–16 |
|---|---|
| R<br>W | — |
| Reset | All zeros |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | — | | | | | | | ECRCCE | ECRCCC | ECRCGE | ECRCGC | First Error Pointer | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

Figure 17-108. PCI Express Advanced Error Capabilities and Control Register

Table 17-104. PCI Express Advanced Error Capabilities and Control Register Field Description

| Bits | Name | Description |
|---|---|---|
| 31–9 | — | Reserved. |
| 8 | ECRCCE | ECRC checking enable. |
| 7 | ECRCCC | ECRC checking capable. |
| 6 | ECRCGE | ECRC generation enable. |
| 5 | ECRCGC | ECRC generation capable. |
| 4–0 | First Error Pointer | The First Error Pointer is a read-only register that identifies the bit position of the first error reported in the Uncorrectable Error Status register. |

### 17.3.10.8 PCI Express Header Log Register—0x11C–0x12B

The PCI Express header log register is shown in Figure 17-109.

Offset 0x11C Access: Read only

| | 31–24 | 23–16 | 15–8 | 7–0 |
|---|---|---|---|---|
| R | Byte 0 | Byte 1 | Byte 2 | Byte 3 |
| W | | | | |
| Reset | All zeros | | | |

Offset 0x120 Access: Read only

| | 31–24 | 23–16 | 15–8 | 7–0 |
|---|---|---|---|---|
| R | Byte 4 | Byte 5 | Byte 6 | Byte 7 |
| W | | | | |
| Reset | All zeros | | | |

Offset 0x124 Access: Read only

| | 31–24 | 23–16 | 15–8 | 7–0 |
|---|---|---|---|---|
| R | Byte 8 | Byte 9 | Byte A | Byte B |
| W | | | | |
| Reset | All zeros | | | |

Offset 0x128 Access: Read only

| | 31–24 | 23–16 | 15–8 | 7–0 |
|---|---|---|---|---|
| R | Byte C | Byte D | Byte E | Byte F |
| W | | | | |
| Reset | All zeros | | | |

**Figure 17-109. PCI Express Header Log Register**

**Table 17-105. PCI Express Header Log Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 127–0 | Header Log | Header of TLP associated with error. |

### 17.3.10.9 PCI Express Root Error Command Register—0x12C

The PCI Express root error command register is shown in Figure 17-110.

Offset 0x12C Access: Read/Write

| | 31–16 |
|---|---|
| R | — |
| W | |
| Reset | All zeros |

| | 15–3 | 2 | 1 | 0 |
|---|---|---|---|---|
| R | — | FERE | NFERE | CERE |
| W | | | | |
| Reset | All zeros | | | |

**Figure 17-110. PCI Express Root Error Command Register**

**Table 17-106. PCI Express Root Error Command Register Field Description**

| Bits | Name | Description |
|---|---|---|
| 31–3 | — | Reserved |
| 2 | FERE | Fatal error reporting enable. |
| 1 | NFERE | Non-fatal error reporting enable |
| 0 | CERE | Correctable error reporting enable |

### 17.3.10.10 PCI Express Root Error Status Register—0x130

The PCI Express root error status register is shown in Figure 17-111.

Offset 0x12C Access: Mixed

| | 31–27 | 26–16 |
|---|---|---|
| R | AEIMN | — |
| W | | |
| Reset | All zeros | |

| | 15–7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| R | — | FEMR | NFEMR | FUF | MEFNFR | EFNFR | MECR | ECR |
| W | | w1c | w1c | w1c | w1c | w1c | w1c | w1c |
| Reset | All zeros | | | | | | | |

**Figure 17-111. PCI Express Root Error Status Register**

**Table 17-107. PCI Express Root Error Command Status Field Description**

| Bits | Name | Description |
|---|---|---|
| 31–27 | AEIMN | Advanced error interrupt message number. |
| 26–7 | — | Reserved |
| 6 | FEMR | Fatal error messages received. |

Table 17-107. PCI Express Root Error Command Status Field Description (continued)

| Bits | Name | Description |
|---|---|---|
| 5 | NFEMR | Non-fatal error messages received. |
| 4 | FUF | First uncorrectable fatal. |
| 3 | MEFNFR | Multiple ERR_FATAL/NONFATAL received. |
| 2 | EFNFR | ERR_FATAL/NONFATAL received. |
| 1 | MECR | Multiple ERR_COR received. |
| 0 | ECR | ERR_COR received. |

### 17.3.10.11 PCI Express Correctable Error Source ID Register—0x134

The PCI Express correctable error source ID register is shown in Figure 17-112.

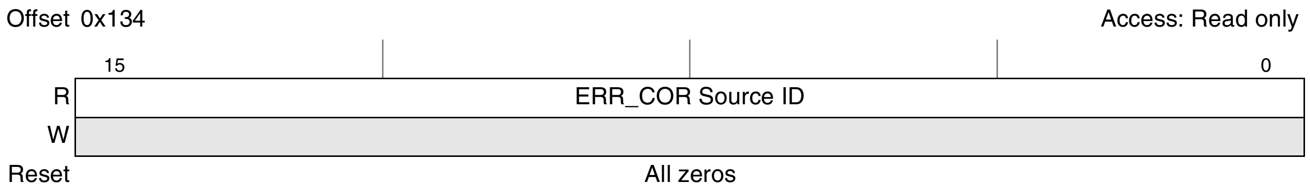


Figure 17-112. PCI Express Correctable Error Source ID Register

Table 17-108. PCI Express Correctable Error Source ID Register Field Description

| Bits | Name | Description |
|---|---|---|
| 15–0 | ERR_COR Source ID | Loaded with the Requestor ID indicated in the received ERR_COR Message when the ERR_COR Received register is not already set. Default value of this field is 0. |

### 17.3.10.12 PCI Express Error Source ID Register—0x136

The PCI Express error source ID register is shown in Figure 17-113.

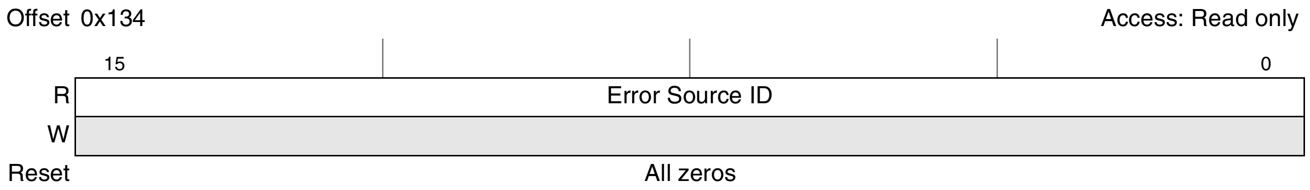


Figure 17-113. PCI Express Correctable Error Source ID Register

Table 17-109. PCI Express Correctable Error Source ID Register Field Description

| Bits | Name | Description |
|---|---|---|
| 15–0 | Error Source ID | ERR_FATAL/NONFATAL source ID |

### 17.3.10.13 LTSSM State Status Register—0x404

The PCI Express link training and status state machine (LTSSM) state status register, shown in Figure 17-114, provides detailed information about link training status. This register is useful for debugging link training failures.

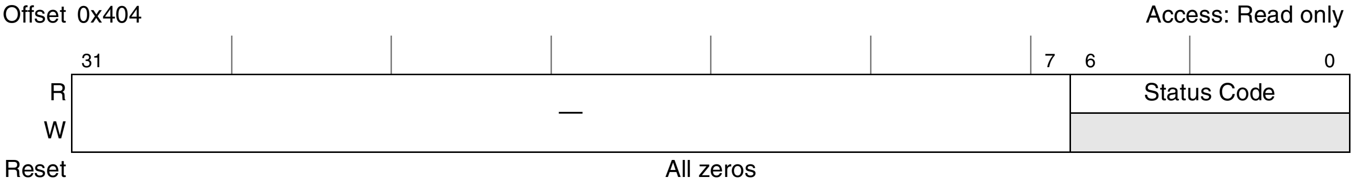


**Figure 17-114. PCI Express LTSSM State Status Register (PEX_LTSSM_STAT)**

The fields of the PCI Express LTSSM state status register are described in Table 17-110.

**Table 17-110. PEX_LTSSM_STAT Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31–7 | — | Reserved |
| 6–0 | Status code | Status code. See Table 17-111 for encodings. |

Table 17-111 provides the encodings for the status code field of the PEX_LTSSM_STAT register.

**Table 17-111. PEX_LTSSM_STAT Status Codes**

| Status Code (Hex) | LTSSM State Description | Status Code (Hex) | LTSSM State Description |
|---|---|---|---|
| 00 | Detect quiet | 27 | TX L0s FTS; RX L0s FTS |
| 01 | Detect active (0) | 28 | L0 to L1 (0) |
| 02 | Detect active (1) | 29 | L0 to L1 (1) |
| 03 | Detect active (2) | 2A | L1 entry |
| 04 | Polling active (0) | 2B | L1 idle (0) |
| 05 | Polling active (1) | 2C | L1 idle (1) |
| 06 | Polling config (0) | 2D | L0 to L2 (0) |
| 07 | Polling config (1) | 2E | L0 to L2 (1) |
| 08 | Polling compliance | 2F | L2 entry |
| 09 | Configuration link width start (0) | 30 | L2 idle (0) |
| 0A | Configuration link width start (1) | 31 | L2 idle (1) |
| 0B | Configuration link width accept (0) | 32 | Recovery lock (0) |
| 0C | Configuration link width accept (1) | 33 | Recovery lock (1) |
| 0D | Configuration lane number wait (0) | 34 | Recovery lock (2) |
| 0E | Configuration lane number wait (1) | 35 | Recovery cfg (0) |
| 0F | Configuration lane number wait (2) | 36 | Recovery cfg (1) |

Table 17-111. PEX_LTSSM_STAT Status Codes (continued)

| Status Code (Hex) | LTSSM State Description | Status Code (Hex) | LTSSM State Description |
|---|---|---|---|
| 10 | Configuration lane number wait (3) | 37 | Recovery idle (0) |
| 11 | Configuration lane number accept | 38 | Recovery idle (1) |
| 12 | Configuration complete (0) | 39 | Recovery to configuration |
| 13 | Configuration complete (1) | 3A | Recovery cfg to configuration |
| 14 | Configuration idle (0) | 3F | L0 no training |
| 15 | Configuration idle (1) | 7F | Detect quiet EI |
| 16 | L0 | 49 | Configuration link width start—RC |
| 17 | TX L0; RX L0s entry | 4A | Configuration link width accept—RC |
| 18 | TX L0; RX L0s idle | 4B | Configuration lane number wait—RC |
| 19 | TX L0; RX L0s fast training sequence (FTS) | 4C | Configuration lane number accept—RC |
| 1A | TX L0s entry (0); RX L0 | 60 | Loopback slave active (0) |
| 1B | TX L0s entry (0); RX L0s idle | 61 | Loopback slave active (1) |
| 1C | TX L0s entry (0); RX L0s FTS | 62 | Loopback slave exit |
| 1D | TX L0s entry (1); RX L0 | 68 | Hot reset (0) |
| 1E | TX L0s entry (1); RX L0s idle | 69 | Hot reset (1) |
| 1F | TX L0s entry (1); RX L0s FTS | 6A | Hot reset (0)—RC |
| 20 | TX L0s idle; RX L0 | 6B | Hot reset (1)—RC |
| 21 | TX L0s idle; RX L0s entry | 75 | Disabled (0) |
| 22 | TX L0s idle; RX L0s idle | 71 | Disabled (1) |
| 23 | TX L0s idle; RX L0s FTS | 72 | Disabled (2) |
| 24 | TX L0s FTS; RX L0 | 73 | Disabled (3) |
| 25 | TX L0s FTS; RX L0s entry | 74 | Disabled (4) |
| 26 | TX L0s FTS; RX L0s idle | 78 | L0 to L1/L2—RC |

### 17.3.10.14 PCI Express Controller Core Clock Ratio Register—0x440

The PCI Express controller core clock ratio register, shown in Figure 17-115, is used to program the ratio of the actual PCI Express controller clock frequency to the default controller core frequency (333 MHz). This is required only when a PCI Express controller clock frequency other than the default 333 MHz has to be used.

Offset 0x440 Access: Read/Write

| Bits | 31–6 | 5–0 |
|---|---|---|
| R<br>W | — | Clock Ratio Numerator |
| Reset | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 1 0 0 0 0 |

**Figure 17-115. PCI Express IP Block Core Clock Ratio Register (PEX_GCLK_RATIO)**

The fields of the PCI Express IP block core clock ratio register are described in Table 17-112.

**Table 17-112. PEX_GCLK_RATIO Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31–6 | — | Reserved |
| 5–0 | Clock Ratio Numerator | The numerator of the ratio of the actual PCI Express controller clock frequency used to the default core clock frequency of 333 MHz. The denominator of the ratio is fixed at 16.<br>The default value of this register is 0x10 (16 decimal), which corresponds to a ratio of 1:1 (or 16/16) |

As an example of programming PEX_GCLK_RATIO, consider the case where the actual PCI Express controller clock is 250 MHz, the ratio of the actual clock to the default clock (333 MHz) is 3:4. that is, the default core clock has to be multiplied by the ratio (3/4, which is equivalent to 12/16). So the register has to be programmed with the decimal numerator value 12 or 0x0000_000C.

### 17.3.10.15 PCI Express Power Management Timer Register—0x450

The PCI Express power management timer register, shown in Figure 17-116, is used to program the time-in values for entering L0s and L1 power management states.

Offset 0x450 Access: Read/Write

| Bits | 31–24 | 23–12 | 11–0 |
|---|---|---|---|
| R<br>W | — | L1_WAIT_PERIOD | L0s_TIME_IN |
| Reset | 0 0 0 0 0 0 0 0 | 0 0 0 1 0 1 0 0 1 1 0 1 | 0 1 1 1 1 1 0 0 1 1 1 0 |

**Figure 17-116. PCI Express Power Management Timer Register (PEX_PM_TIMER)**

The fields of the PCI Express power management timer register are described in Table 17-113.

**Table 17-113. PEX_PM_TIMER Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31–24 | — | Reserved |
| 23–12 | L1_WAIT_PERIOD | Wait period (in PCI Express controller core clock cycles) before entering L1 power state after all functions are in a non-D0 power state. The value is calculated as:<br>Time (in µsec) × PCI Express controller core clock frequency (in MHz)<br>The time value must be less than 2 µsec; the default value (0x14D) is 1 µsec for the default clock frequency of 333 MHz. |
| 11–0 | L0s_TIME_IN | Time in value (in PCI Express controller core clock cycles) for entering L0s power state. The value is calculated as:<br>Time (in µsec) × PCI Express controller core clock frequency (in MHz)<br>The maximum time value is 7 µsec; the default value (0x7CE) is 6 µsec for the default clock frequency of 333 MHz. |

## 17.3.10.16 PCI Express PME Time-Out Register (EP-Mode Only)—0x454

The PCI Express PME time-out register, shown in Figure 17-117, is used to program the time-out value that the controller uses before re-sending a PME message to the host. If PME is requested by a function and the host does not clear the associated PME_STAT bit even after this time-out has expired, the PME message is sent again to the host by the PCI Express controller. This register is supported only for EP mode.

Offset 0x454 (EP-mode only) Access: Read/Write

| | 31–26 | 25–0 |
|---|---|---|
| R | — | PME_TIMEOUT |
| W | | |
| Reset | 0 0 0 0 0 0 | 0 1 1 1 1 1 1 1 0 0 0 0 0 1 1 1 1 0 0 0 1 0 0 0 0 0 |

**Figure 17-117. PCI Express PME Time-Out Register (PEX_PME_TIMEOUT)**

The fields of the PCI Express PME time-out register are described in Table 17-113.

**Table 17-114. PEX_PME_TIMEOUT Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31–26 | — | Reserved |
| 25–0 | PME_TIMEOUT | The PME time-out value specifies the interval before PME messages are resent by the controller, provided the PME_STAT bit in the PCI Express power management status and control register (offset 0x48) is not cleared by the host. The value for PME_TIMEOUT is specified in terms of PCI Express controller core clock cycles. The value is calculated as:<br>Time (in µsec) × PCI Express controller core clock frequency (in MHz)<br>The minimum time value is 100 msec; the default value (0x1FC1E20) is 100 msec for the default clock frequency of 333 MHz. |

### 17.3.10.17 PCI Express Subsystem Vendor ID Update Register (EP Mode Only)—0x478

The PCI Express subsystem vendor ID update register, shown in Figure 17-118, is used to set the values for the Subsystem ID and Subsystem Vendor ID registers in the Type 0 configuration header.

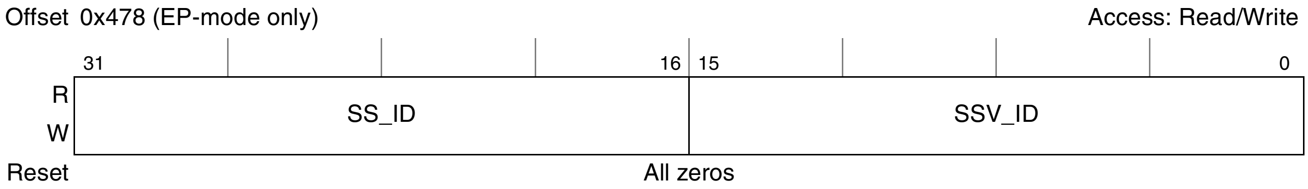


**Figure 17-118. PCI Express Subsystem Vendor ID Update Register (PEX_SSVID_UPDATE)**

The fields of the PCI Express subsystem vendor ID update register are described in Table 17-115.

**Table 17-115. PEX_SSVID_UPDATE Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31–16 | SS_ID | Subsystem ID [15–0] value |
| 15–0 | SSV_ID | Subsystem vendor ID [15–0] value |

When used as an endpoint, the controller’s initialization software programs the desired subsystem ID and subsystem vendor ID values in PEX_SSVID_UPDATE before setting the CFG_READY bit in the PEX_CFG_READY register (see Section 17.3.10.18, “Configuration Ready Register—0x4B0”). That way, when the host begins system enumeration, the correct values are present in the Type 0 configuration header.

### 17.3.10.18 Configuration Ready Register—0x4B0

The PCI Express configuration ready register, shown in Figure 17-119, is used to indicate configuration complete status to the transaction layer. The transaction layer handles configuration requests from external hosts only after the CFG_READY bit is set. All the configuration requests received from external hosts before the CFG_READY bit is set are completed with configuration request retry status (CRS). The CFG_READY bit in this register should be set after all relevant configuration registers have been programmed. This makes sure the external host reads the correct capability advertisements during enumeration.

Note that the state of PEX_CFG_READY[CFG_READY] is dependent upon the POR configuration setting described in Section 17.5.1, “Boot Mode and Inbound Configuration Transactions.”

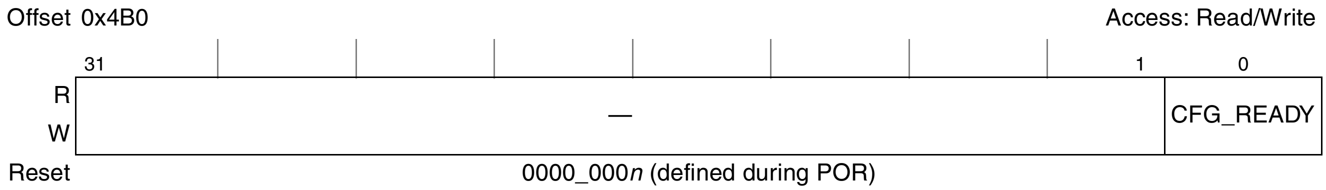


**Figure 17-119. PCI Express Configuration Ready Register (PEX_CFG_READY)**

The fields of the PCI Express configuration ready register are described in Table 17-116.

**Table 17-116. PEX_CFG_READY Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31–1 | — | Reserved |
| 0 | CFG_READY | Configuration ready<br>1 The transaction layer accepts inbound configuration requests.<br>0 The transaction layer responds to all inbound configuration requests with retry (CRS)<br>Note that the reset state of this bit is determined during POR. |

## 17.3.10.19 PME_To_Ack Timeout Register (RC-Mode Only)—0x590

The PCI Express PME_To_Ack timeout register, shown in Figure 17-120, is used to program the timeout value for a PME_To_Ack message response in terms of PCI Express controller core clock cycles.

Offset 0x590 (RC-mode only) Access: Read/Write

| | 31–22 | 21–0 |
|---|---|---|
| R<br>W | — | PME_TO_ACK_TIMEOUT |
| Reset | 1 0 0 0 0 0 0 0 0 0 | 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 |

**Figure 17-120. PCI Express PME_To_Ack Timeout Register (PEX_PME_TO_ACK_TOR)**

The fields of the PCI Express PME_To_Ack timeout register are described in Table 17-117.

**Table 17-117. PEX_PME_TO_ACK_TOR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31–22 | — | Reserved |
| 21–0 | PME_TO_ACK<br>_<br>TIMEOUT | After a PME_Turn_Off message is broadcast by the RC, the power management module waits for the duration of the PME_To_Ack timeout interval to receive a PME_To_Ack message from the downstream device. If the Ack message is not received within this interval, the power manager indicates that it is safe to switch off power, since timeout has occurred.<br>The value is calculated as:<br>Time (in µsec) × PCI Express controller core clock frequency (in MHz)<br>The recommended timeout duration is 1 msec to 10 msec to make sure that the downstream devices get enough time to prepare for power-off condition. |

## 17.3.10.20 Secondary Status Interrupt Mask Register (RC-Mode Only)—0x5A0

The PCI Express secondary status interrupt mask register, shown in Figure 17-121, can be used to disable sideband interrupt generation when error bits in the PCI Express secondary status register are set. See Section 17.3.8.3.8, “PCI Express Secondary Status Register—0x1E,” for more information. By default, interrupt generation due to secondary status errors is disabled.

Offset 0x5A0 (RC-mode only) Access: Mixed

| | 31–6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| R | — | M_DPE | M_SSE | M_RMA | M_RTA | M_STA | M_MDPE |
| W | | | | | | | |
| Reset | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 1 | 1 | 1 | 1 | 1 | 1 |

**Figure 17-121. PCI Express PCI Interrupt Mask Register (PEX_SS_INTR_MASK)**

The fields of the PCI Express secondary status interrupt mask register are described in Table 17-118.

**Table 17-118. PEX_SS_INTR_MASK Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31–6 | — | Reserved |
| 5 | M_DPE | Mask detected parity error |
| 4 | M_SSE | Mask signaled system error |
| 3 | M_RMA | Mask received master abort |
| 2 | M_RTA | Mask received target abort |
| 1 | M_STA | Mask signaled target abort |
| 0 | M_MDPE | Mask master data parity error |

## 17.4 Functional Description

The PCI Express protocol relies on a requestor/completer relationship where one device requests that some desired action be performed by some target device and the target device completes the task and responds. Usually the requests and responses occur through a network of links, but to the requestor and to the completer, the intermediate components are transparent.

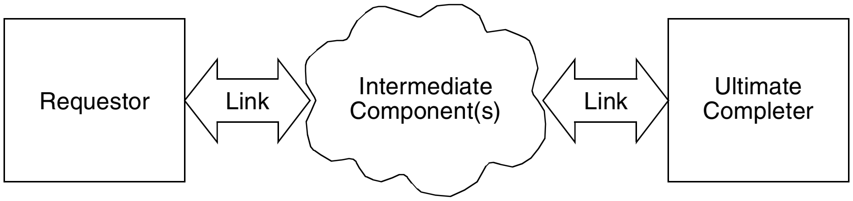


**Figure 17-122. Requestor/Completer Relationship**

Each PCI Express device is divided into two halves-transmit (TX) and receive (RX), and each of these halves is further divided into three layers—transaction, data link, and physical—as shown in Figure 17-123.

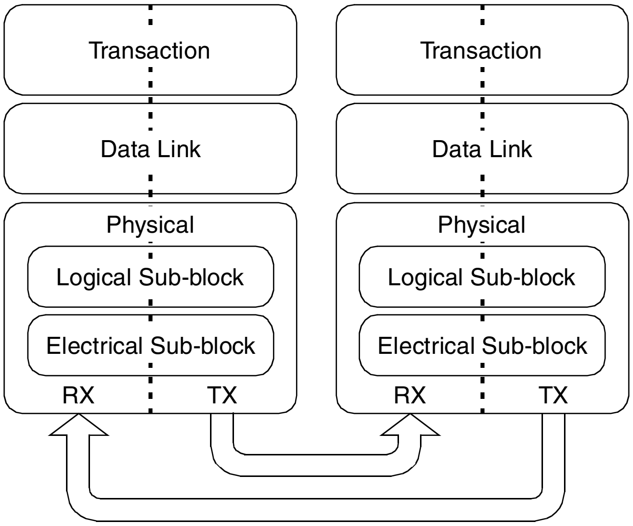


**Figure 17-123. PCI Express High-Level Layering**

Packets are formed in the transaction layer (TLPs) and data link layer (DLLPs), and each subsequent layer adds the necessary encodings and framing—as shown in Figure 17-124. As packets are received, they are decoded and processed by the same layers but in reverse order, so they may be processed by the layer or by the device application software.

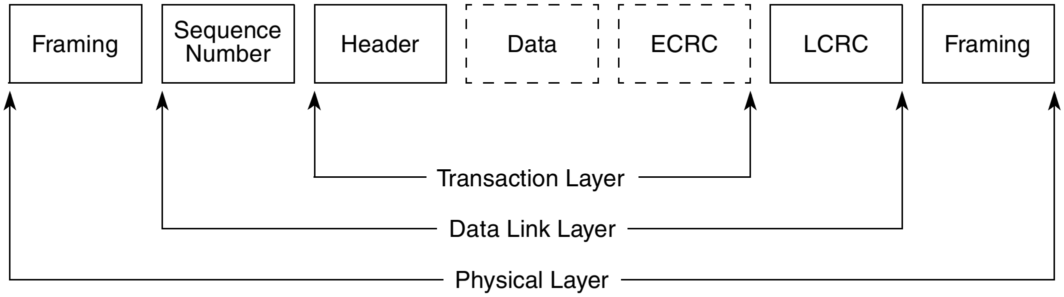


**Figure 17-124. PCI Express Packet Flow**

## 17.4.1 Architecture

This section describes implementation details of the PCI Express controller.

### 17.4.1.1 PCI Express Transactions

Table 17-119 contains the list of transactions that the PCI Express controller supports as an initiator and a target.

Table 17-119. PCI Express Transactions

| PCI Express Transaction | Supported as an Initiator | Supported as a Target | Definition |
|---|---|---|---|
| Mrd | Yes | Yes | Memory Read Request |
| MRdLk | No | No | Memory Read Lock. As a target, CplLk with UR status is returned. |
| MWr | Yes | Yes | Memory Write Request to memory-mapped PCI-Express space |
| IORd | Yes (RC only) | No | I/O Read request. As a target, Cpl with UR status is returned. |
| IOWr | Yes (RC only) | No | I/O Write Request. As a target, Cpl with UR status is returned. |
| CfgRd0 | Yes (RC only | Yes | Configuration Read Type 0 |
| CfgWr0 | Yes (RC only) | Yes | Configuration Write Type 0 |
| CfgRd1 | Yes (RC only) | No | Configuration Read Type 1. As a target, Cpl with UR status is returned. |
| CfgWr1 | Yes (RC only) | No | Configuration Write Type 1. As a target, Cpl with UR status is returned. |
| Msg | Yes | Yes | Message Request |
| MsgD | Yes (RC only) | Yes (EP only) | Message Request with Data payload. Note that Set_Slot_Power_Limit is the only message with data that is supported and then only when the controller is an initiator and in RC mode or a target and in EP mode. |
| Cpl | Yes | Yes | Completion without Data |
| CplD | Yes | Yes | Completion with Data |
| CplLk | No | Yes | Completion for Locked Memory Read without Data. The only time that CplLk is returned with UR status is when the controller receives a MRdLk command. |
| CplDLk | No | No | Completion for Locked Memory Read with Data |

### 17.4.1.2 Byte Ordering

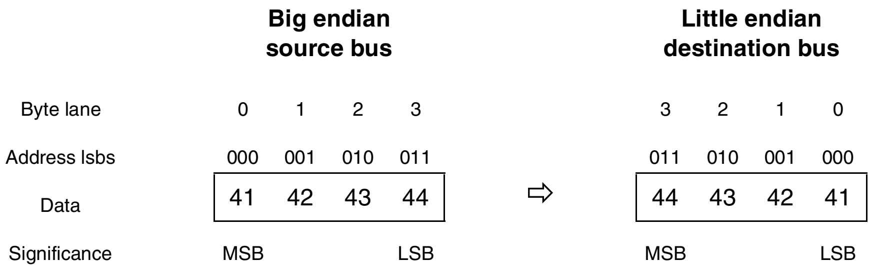
Whenever data must cross a bridge between two busses, the byte ordering of data on the source and destination buses must be considered. The internal platform bus of this device is inherently big endian and the PCI Express bus interface is inherently little endian.

There are two methods to handle ordering of data as it crosses a bridge—address invariance and data invariance. Address invariance preserves the addressing of bytes within a scalar data element, but not the relative significance of the bytes within that scalar. Conversely, data invariance preserves the relative significance of bytes within a scalar, but not the addressing of the individual bytes that make up a scalar.

This device uses address invariance as its byte ordering policy.

As stated above, address invariance preserves the byte address of each byte on an I/O interface as it is placed in memory or moved into a register. This policy can have the effect of reversing the significance order of bytes (most significant to least significant and vice versa), but it has the benefit of preserving the format of general data structures. Provided that software is aware of the endianness and format of the data structure, it can correctly interpret the data on either side of the bridge.

Figure 17-125 shows the transfer of a 4-byte scalar, 0x4142_4344, from a big endian source across an address invariant bridge to a little endian destination.

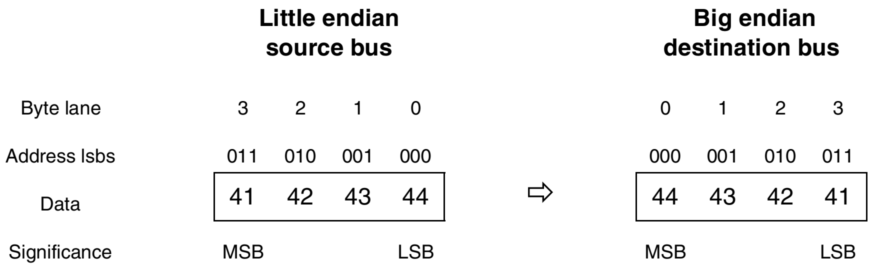


| | Big endian source bus | | | | | Little endian destination bus | | | |
|---|---|---|---|---|---|---|---|---|---|
| Byte lane | 0 | 1 | 2 | 3 | | 3 | 2 | 1 | 0 |
| Address lsbs | 000 | 001 | 010 | 011 | | 011 | 010 | 001 | 000 |
| Data | 41 | 42 | 43 | 44 | ⇨ | 44 | 43 | 42 | 41 |
| Significance | MSB | | | LSB | | MSB | | | LSB |

Figure 17-125. Address Invariant Byte Ordering—4 bytes Outbound

Note that although the significance of the bytes within the scalar have changed, the address of the individual bytes that make up the scalar have not changed. As long as software is aware that the source of the data used a big endian format, the data can be interpreted correctly.

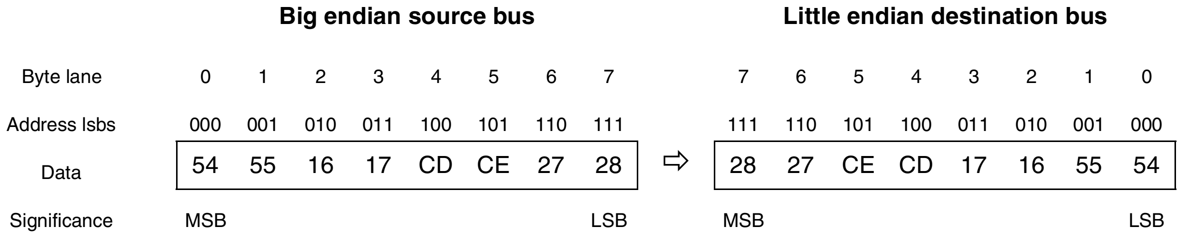
Figure 17-127 shows data flowing the other way, from a little endian source to a big endian destination.



| | Little endian source bus | | | | | Big endian destination bus | | | |
|---|---|---|---|---|---|---|---|---|---|
| Byte lane | 3 | 2 | 1 | 0 | | 0 | 1 | 2 | 3 |
| Address lsbs | 011 | 010 | 001 | 000 | | 000 | 001 | 010 | 011 |
| Data | 41 | 42 | 43 | 44 | ⇨ | 44 | 43 | 42 | 41 |
| Significance | MSB | | | LSB | | MSB | | | LSB |

Figure 17-126. Address Invariant Byte Ordering—4 bytes Inbound

Figure 17-127 shows an outbound transfer of an 8-byte scalar, 0x5455_1617_CDCE_2728, using address invariance.



| | Big endian source bus | | | | | | | | | Little endian destination bus | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Byte lane | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Address lsbs | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 | | 111 | 110 | 101 | 100 | 011 | 010 | 001 | 000 |
| Data | 54 | 55 | 16 | 17 | CD | CE | 27 | 28 | ⇨ | 28 | 27 | CE | CD | 17 | 16 | 55 | 54 |
| Significance | MSB | | | | | | | LSB | | MSB | | | | | | | LSB |

Figure 17-127. Address Invariant Byte Ordering—8 bytes Outbound

Figure 17-128 shows an inbound transfer of a 2-byte scalar, 0x5837, using address invariance.

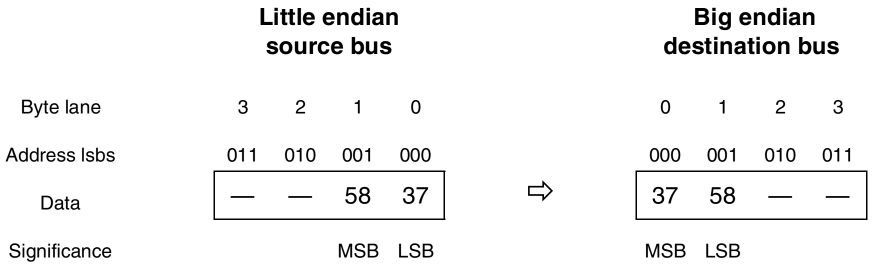


**Figure 17-128. Address Invariant Byte Ordering—2 bytes Inbound**

Note that in all of these examples, the original addresses of the individual bytes within the scalars (as created by the source) have been preserved.

### 17.4.1.2.1 Byte Order for Configuration Transactions

All internal memory-mapped registers in the CCSR space use big endian byte ordering. However, the PCI Express specification defines PCI Express configuration registers as little endian. All accesses to the PCI Express configuration port, PEX_CONFIG_DATA, including the those targeting the internal PCI Express configuration registers, use the address invariance policy as shown in Figure 17-129. Therefore, software must access PEX_CONFIG_DATA with little-endian formatted data—either using the **lwbrx**/**stwbrx** instructions or by manipulating the data before writing to and after reading from PEX_CONFIG_DATA.

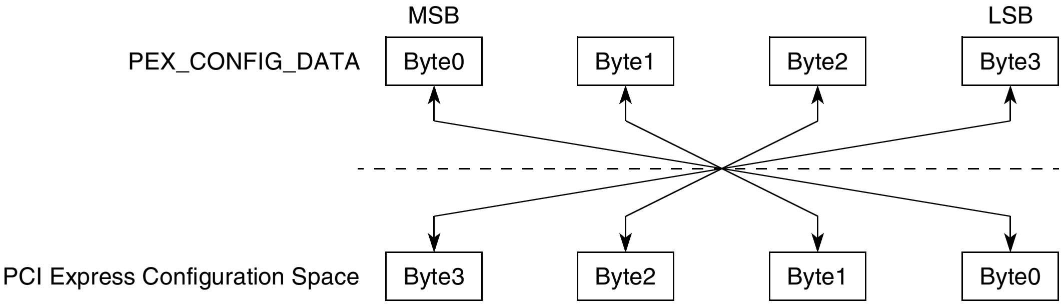


**Figure 17-129. PEX_CONFIG_DATA Byte Ordering**

## 17.4.1.3 Lane Reversal

The PCI Express link supports lane reversal. Table 17-120 describes the supported configurations.

**Table 17-120. Lane Assignment With and Without Lane Reversal**

| Link Configuration | Lane 0 | Lane 1 | Lane 2 | Lane 3 | Lane 4 | Lane 5 | Lane 6 | Lane 7 |
|---|---|---|---|---|---|---|---|---|
| x8 link without lane reversal | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| x4 link without lane reversal | 0 | 1 | 2 | 3 | — | — | — | — |
| x2 link without lane reversal | 0 | 1 | — | — | — | — | — | — |
| x1 link without lane reversal | 0 | — | — | — | — | — | — | — |
| x8 link with lane reversal | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Table 17-120. Lane Assignment With and Without Lane Reversal (continued)

| Link Configuration | Lane 0 | Lane 1 | Lane 2 | Lane 3 | Lane 4 | Lane 5 | Lane 6 | Lane 7 |
|---|---|---|---|---|---|---|---|---|
| x4 link with lane reversal | — | — | — | — | 3 | 2 | 1 | 0 |
| x2 link with lane reversal | — | — | — | — | — | — | 1 | 0 |
| x1 link with lane reversal | — | — | — | — | — | — | — | 0 |

**Note:** The numbers shown in this table (0–7) are the lane numbers assigned to each lane as a result of link initialization and configuration.
— indicates that the lane is not part of the configured link.

Note that lane reversal is only effective for devices that use the full 8 lanes. That is, if a x4 device is connected to lanes 0–3 and the link training fails without lane reversal, the lane reversal causes the link to attempt connection on lanes 7–4 which would be impossible.

### 17.4.1.4 Transaction Ordering Rules

In general, transactions are serviced in the order that they are received. However, transactions can be reordered as they are sent due to a stalled condition such as a full internal buffer. The following are the ordering rules for sending the next outstanding request:

- A posted request can and bypasses all other transactions except another posted request.
- A completion can and only bypasses non-posted. It can and bypasses posted requests only if the relaxed ordering (RO) bit is set.
- A non-posted request cannot bypass posted or other non-posted requests, but it can bypass a completion if the relaxed ordering (RO) bit is set.

### 17.4.1.5 Memory Space Addressing

A PCI Express memory transaction can address a 32- or 64-bit memory space. The FMT[0] field in the PCI Express TLP header for a 32-bit address packet is 0; a 64-bit address packet has a FMT[0] = 1. The PCI Express TLP header for a memory read transaction has TYPE[4:0] = 00000 and FMT[1] = 0. A memory write transaction has TYPE[4:0] = 00000 and FMT[1] = 1. As an initiator, the controller is capable of sending 32- or 64-bit memory packets. Any transaction from the internal platform that (after passing through the translation mechanism) has a translated address greater than 4G is sent as a 64-bit memory packet. Otherwise, a 32-bit memory packet is sent. As a target device, the controller is capable of decoding 32- or 64-bit memory packets. This is done through two 32-bit inbound windows and two 64-bit inbound windows. All inbound addresses are translated to 36-bit internal platform addresses.

### 17.4.1.6 I/O Space Addressing

The controller does not support I/O transactions as a target. As an initiator, the controller can send I/O transactions in RC mode only. This can be done by programming one of the outbound translation window’s attribute to send I/O transactions. All I/O transactions only access 32-bit address I/O space. The PCI Express TLP header for an I/O read transaction has TYPE[4:0] = 00010 and FMT[1] = 0. The PCI Express TLP header for an I/O write transaction has TYPE[4:0] = 00010 and FMT[1] = 1.

### 17.4.1.7 Configuration Space Addressing

As an initiator, the controller supports both type 0 and type 1 configuration cycles when configured in RC mode. There are two methods of generating a configuration transaction; refer to Section 17.3.7, "PCI Express Configuration Space Access," for more information. A configuration transaction can hit into the controller's internal configuration space, it can be sent out on the PCI Express link, or it can be internally terminated. The PCI Express TLP header for a type 0 configuration read transaction has TYPE[4:0] = 00100 and FMT[1] = 0; the PCI Express TLP header for a type 0 configuration write transaction has TYPE[4:0] = 00100 and FMT[1] = 1. The PCI Express TLP header for a type 1 configuration read transaction has TYPE[4:0] = 00101 and FMT[1] = 0; the PCI Express TLP header for a type 1 configuration write transaction has TYPE[4:0] = 00101 and FMT[1] = 1. Note that all configuration transactions sent on PCI Express require a response regardless whether they are read or a write configuration transactions.

The controller does not generate configuration transactions in EP mode. Only inbound configuration transactions are supported in EP mode.

### 17.4.1.8 Serialization of Configuration and I/O Writes

Configuration and I/O writes are serialized by the controller. The logic after issuing a configuration write or IO write does not issue any new transactions until the outstanding configuration or I/O write is finished. This means that an acknowledgement packet from the link partner in the form of a CpL TLP packet must be seen or the transaction has timed out. If the CpL packet contains a CRS status, then the logic re-issues the configuration write transaction. It keeps retrying the request until either a status other than CRS is returned or the transaction times out.

Note that it is possible for outbound configuration read request to be requeued and be placed at the end of the request queue due to CRS condition.

### 17.4.1.9 Messages

Software message generation is supported in both RC and EP modes.

#### 17.4.1.9.1 Outbound ATMU Message Generation

Software can choose to send a message by programming PEXOWAR*n*[WTT] = 0x5. A message is sent by writing a 4-byte transaction in big-endian format that hits in an outbound window configured to send messages.

Part of the 4-byte data is used to store information such as message code and routing information. Table 17-121 describes the message data format.

**Table 17-121. Internal Platform (OCeaN) Message Data Format**

| Bits | Name | Reset Value | Description |
|---|---|---|---|
| 0–15 | — | — | Reserved |
| 16–18 | Routing | x | Routing mechanism. Contains the message's routing information |

Table 17-121. Internal Platform (OCeaN) Message Data Format

| Bits | Name | Reset Value | Description |
|---|---|---|---|
| 19–23 | — | — | Reserved |
| 24–31 | Code | x | Message code. Contains the actual message type to be sent. |

In addition to the outbound ATMU, the PEX PM Command register also provides the capability to send PME_Turn_Off message or PM_PME message by setting bits 31 or 29. See Section 17.3.3.4, “PCI Express Power Management Command Register (PEX_PMCR),” on page 17-18 for more information.

Table 17-122 provides a complete list of supported outbound messages depending on whether RC or EP is configured.

Table 17-122. PCI Express ATMU Outbound Messages

| Name | Code[7:0] | Routing[2:0] | RC | EP | Description |
|---|---|---|---|---|---|
| PM_Active_State_Nak | 0001 0100 | 100 | Yes | N/A | Terminate at receiver |
| PM_PME | 0001 1000 | 000 | N/A | Yes | Sent Upstream by PME-requesting component |
| PME_Turn_Off | 0001 1001 | 011 | Yes | N/A | Broadcast Downstream |
| PM_TO_Ack | 0001 1011 | 101 | N/A | Yes | Sent Upstream by Endpoint |
| ERR_COR | 0011 0000 | 000 | N/A | Yes | Sent by component when it detects a correctable error |
| ERR_NONFATAL | 0011 0001 | 000 | N/A | Yes | Sent by component when it detects a Non-fatal, uncorrectable error |
| ERR_FATAL | 0011 0011 | 000 | N/A | Yes | Sent by component when it detects a Fatal, uncorrectable error |
| Unlock | 0000 0000 | 000 | No | N/A | Not supported |
| Set_Slot_Power_Limit | 0101 0000 | 100 | Yes | N/A | Set Slot Power Limit in Upstream Port |
| Vendor_Defined Type 0 | 0111 1110 | | No | No | Not supported |
| Vendor_Defined Type 1 | 0111 1111 | | No | No | Not supported |
| Attention_Indicator_On | 0100 0001 | 100 | Yes | N/A | Hot-plug message |
| Attention_Indicator_Blink | 0100 0011 | 100 | Yes | N/A | Hot-plug message |
| Attention_Indicator_Off | 0100 0000 | 100 | Yes | N/A | Hot-plug message |
| Power_Indicator_On | 0100 0101 | 100 | Yes | N/A | Hot-plug message |
| Power_Indicator_Blink | 0100 0111 | 100 | Yes | N/A | Hot-plug message |
| Power_Indicator_Off | 0100 0100 | 100 | Yes | N/A | Hot-plug message |
| Attention_Button_Pressed | 0100 1000 | 100 | Yes | N/A | Hot-plug message |

#### 17.4.1.9.2 Inbound Messages

Table 17-123 provides a complete list of supported inbound messages in RC mode.

Table 17-123. PCI Express RC Inbound Message Handling

| Name | Code[7:0] | Routing[2:0] | Action |
|---|---|---|---|
| Assert_INTA | 0010 0000 | 100 | Send to PIC |
| Assert_INTB | 0010 0001 | 100 | Send to PIC |
| Assert_INTC | 0010 0010 | 100 | Send to PIC |
| Assert_INTD | 0010 0011 | 100 | Send to PIC |
| De-assert_INTA | 0010 0100 | 100 | Send to PIC |
| De-assert_INTB | 0010 0101 | 100 | Send to PIC |
| De-assert_INTC | 0010 0110 | 100 | Send to PIC |
| De-assert_INTD | 0010 0111 | 100 | Send to PIC |
| PM_Active_State_Nak | 0001 0100 | 100 | No action taken |
| PM_PME | 0001 1000 | 000 | Generate interrupt to PIC if enabled |
| PME_Turn_Off | 0001 1001 | 011 | No action taken |
| PME_TO_Ack | 0001 1011 | 101 | Set PEX_PME_MES_DR[ENL23] bit and generate interrupt to PIC if enabled |
| ERR_COR | 0011 0000 | 000 | Generate interrupt to PIC if enabled |
| ERR_NONFATAL | 0011 0001 | 000 | Generate interrupt to PIC if enabled |
| ERR_FATAL | 0011 0011 | 000 | Generate interrupt to PIC if enabled |
| Unlock | 0000 0000 | 000 | No action taken |
| Set_Slot_Power_Limit | 0101 0000 | 100 | No action taken |
| Vendor_Defined Type 0 | 0111 1110 |  | No action taken |
| Vendor_Defined Type 1 | 0111 1111 |  | No action taken |
| Attention_Indicator_On | 0100 0001 | 100 | No action taken |
| Attention_Indicator_Blink | 0100 0011 | 100 | No action taken |
| Attention_Indicator_Off | 0100 0000 | 100 | No action taken |
| Power_Indicator_On | 0100 0101 | 100 | No action taken |
| Power_Indicator_Blink | 0100 0111 | 100 | No action taken |
| Power_Indicator_Off | 0100 0100 | 100 | No action taken |
| Attention_Button_Pressed | 0100 1000 | 100 | Set PEX_PME_MES_DR[ABP] bit and send interrupt if enabled. |

Table 17-124 provides a complete list of supported inbound messages in EP mode.

Table 17-124. PCI Express EP Inbound Message Handling

| Name | Code[7:0] | Routing[2:0] | Action |
|---|---|---|---|
| Assert_INTA | 0010 0000 | 100 | No action taken |
| Assert_INTB | 0010 0001 | 100 | No action taken |
| Assert_INTC | 0010 0010 | 100 | No action taken |
| Assert_INTD | 0010 0011 | 100 | No action taken |
| Deassert_INTA | 0010 0100 | 100 | No action taken |
| Deassert_INTB | 0010 0101 | 100 | No action taken |
| Deassert_INTC | 0010 0110 | 100 | No action taken |
| Deassert_INTD | 0010 0111 | 100 | No action taken |
| PM_Active_State_Nak | 0001 0100 | 100 | No action taken |
| PM_PME | 0001 1000 | 000 | No action taken |
| PME_Turn_Off | 0001 1001 | 011 | Set PEX_PME_MES_DR[PTO] bit. Send interrupt if enabled. |
| PM_TO_Ack | 0001 1011 | 101 | No action taken |
| ERR_COR | 0011 0000 | 000 | No action taken |
| ERR_NONFATAL | 0011 0001 | 000 | No action taken |
| ERR_FATAL | 0011 0011 | 000 | No action taken |
| Unlock | 0000 0000 | 000 | No action taken |
| Set_Slot_Power_Limit | 0101 0000 | 100 | Update power value in PCI Express device capability register in configuration space. |
| Vendor_Defined Type 0 | 0111 1110 | | No action taken |
| Vendor_Defined Type 1 | 0111 1111 | | No action taken |
| Attention_Indicator_On | 0100 0001 | 100 | Set PEX_PME_MES_DR[AION] bit. Send interrupt if enabled. |
| Attention_Indicator_Blink | 0100 0011 | 100 | Set PEX_PME_MES_DR[AIB] bit. Send interrupt if enabled. |
| Attention_Indicator_Off | 0100 0000 | 100 | Set PEX_PME_MES_DR[AIOF] bit. Send interrupt if enabled. |
| Power_Indicator_On | 0100 0101 | 100 | Set PEX_PME_MES_DR[PION] bit. Send interrupt if enabled. |
| Power_Indicator_Blink | 0100 0111 | 100 | Set PEX_PME_MES_DR[PIB] bit. Send interrupt if enabled. |
| Power_Indicator_Off | 0100 0100 | 100 | Set PEX_PME_MES_DR[PIOF] bit. Send interrupt if enabled. |
| Attention_Button_Pressed | 0100 1000 | 100 | No action taken |

### 17.4.1.10 Error Handling

The PCI Express specification classifies errors as correctable and uncorrectable. Correctable errors result in degraded performance, but uncorrectable errors generally result in functional failures. As shown in Figure 17-130 uncorrectable errors can further be classified as fatal or non-fatal.

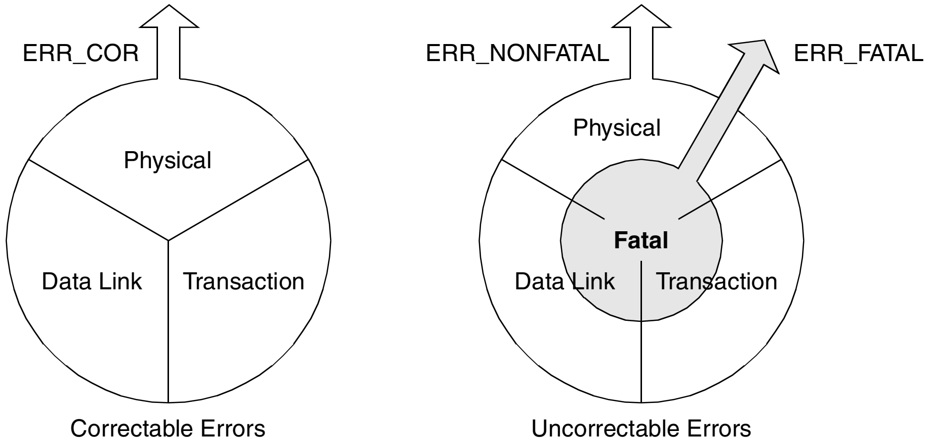
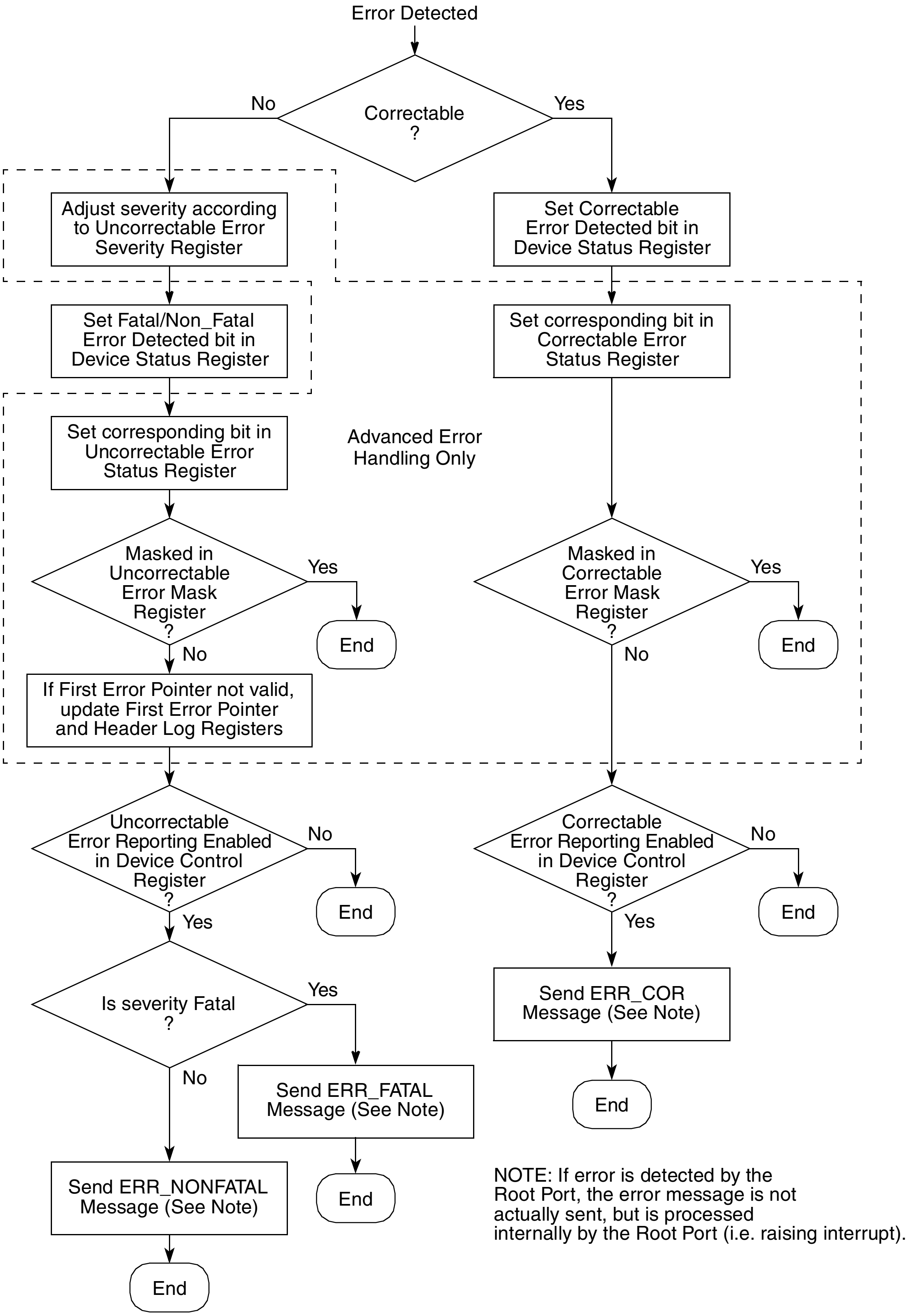


**Figure 17-130. PCI Express Error Classification**

#### 17.4.1.10.1 PCI Express Error Logging and Signaling

Figure 17-131 shows the PCI Express-defined sequence of operations related to signaling and logging of errors detected by a device. Note that the PCI Express controller on this device supports the advanced error handling capabilities shown within the dotted lines.


Error Detected
Correctable ?
No
Yes
Adjust severity according to Uncorrectable Error Severity Register
Set Correctable Error Detected bit in Device Status Register
Set Fatal/Non_Fatal Error Detected bit in Device Status Register
Set corresponding bit in Correctable Error Status Register
Set corresponding bit in Uncorrectable Error Status Register
Advanced Error Handling Only
Masked in Uncorrectable Error Mask Register ?
Yes
End
No
Masked in Correctable Error Mask Register ?
Yes
End
No
If First Error Pointer not valid, update First Error Pointer and Header Log Registers
Uncorrectable Error Reporting Enabled in Device Control Register ?
No
End
Yes
Correctable Error Reporting Enabled in Device Control Register ?
No
End
Yes
Is severity Fatal ?
Yes
No
Send ERR_FATAL Message (See Note)
End
Send ERR_COR Message (See Note)
End
Send ERR_NONFATAL Message (See Note)
End

NOTE: If error is detected by the Root Port, the error message is not actually sent, but is processed internally by the Root Port (i.e. raising interrupt).

**Figure 17-131. PCI Express Device Error Signaling Flowchart**

#### 17.4.1.10.2 PCI Express Controller Internal Interrupt Sources

Table 17-125 describes the sources of the PCI Express controller internal interrupt to the PIC and the preconditions for signaling the interrupt.

Table 17-125. PCI Express Internal Controller Interrupt Sources

| Status Register Bit | Preconditions |
|---|---|
| Any bit in PEX_PME_MES_DR set | The corresponding interrupt enable bits must be set in PEX_PME_MES_IER |
| Any bit in PEX_ERR_DR set | The corresponding interrupt enable bits must be set in PEX_ERR_EN. |
| PCI Express Root Status Register[16] (PME status) is set | PCI Express Root Control Register [3] (PME interrupt enable) is set |
| PCI Express Root Error Status Register[6] (fatal error messages received) is set | PCI Express Root Error Command Register [2] (fatal error reporting enable) is set<br>or<br>PCI Express Root Control Register [2] (system error on fatal error enable) is set |
| PCI Express Root Error Status Register [5] (non-fatal error messages received) is set | PCI Express Root Error Command Register [1] (non-fatal error reporting enable) is set<br>or<br>PCI Express Root Control Register [1] (system error on non-fatal error enable) is set |
| PCI Express Root Error Status Register[0] (correctable error messages received) is set | PCI Express Root Error Command Register[0] (correctable error reporting enable) is set<br>or<br>PCI Express Root Control Register[0] (system error on correctable error enable) is set. |
| Any correctable error status bit in PCI Express Correctable Error Status Register is set | The corresponding error mask bit in PCI Express Correctable Error Mask Register is clear and PCI Express Root Error Command Register[0] (correctable error reporting enable) is set |
| Any fatal uncorrectable error status bit in PCI Express Uncorrectable Error Status Register is set. (The corresponding error is classified as fatal based on the severity setting in PCI Express Uncorrectable Error Severity Register.) | The corresponding error mask bit in PCI Express Uncorrectable Error Mask Register is clear and either PCI Express Device Control Register[2] (fatal error reporting) is set or PCI Express Command Register[8] (SERR) is set. |
| Any non-fatal uncorrectable error status bit in PCI Express Uncorrectable Error Status Register is set. (The corresponding error is classified as non-fatal based on the severity setting in PCI Express Uncorrectable Error Severity Register.) | The corresponding error mask bit in PCI Express Uncorrectable Error Mask Register is clear<br>and<br>either PCI Express Device Control Register[1] (non-fatal error reporting) is set or PCI Express Command Register[8] (SERR) is set. |
| PCI Express Secondary Status Register[8] (master data parity error) is set. | PCI Express Secondary Status Interrupt Mask Register[0] (mask master data parity error) is cleared and PCI Express Command Register[6] (parity error response) is set. |
| PCI Express Secondary Status Register[11] (signaled target abort) is set | PCI Express Secondary Status Interrupt Mask Register[1] (mask signaled target abort) is cleared. |
| PCI Express Secondary Status Register[12] (received target abort) is set | PCI Express Secondary Status Interrupt Mask Register[2] (mask received target abort) is cleared. |

Table 17-125. PCI Express Internal Controller Interrupt Sources (continued)

| Status Register Bit | Preconditions |
|---|---|
| PCI Express Secondary Status Register[13] (received master abort) is set | PCI Express Secondary Status Interrupt Mask Register[3] (mask received master abort) is cleared. |
| PCI Express Secondary Status Register[14] (signaled system error) is set. | PCI Express Secondary Status Interrupt Mask Register[4] (mask signaled system error) is cleared. |
| PCI Express Secondary Status Register[15] (detected parity error) is set | PCI Express Secondary Status Interrupt Mask Register[5] (mask detected parity error) is cleared. |
| PCI Express Slot Status Register[0] (attention button pressed) is set | PCI Express Slot Control Register[0] (attention button pressed enable) is set<br>and<br>PCI Express Slot Control Register[5] (hot plug interrupt enable) is set<br>and<br>either PCI Express PM Control Register[1–0] = 00 (the function power state is D0) or PCI Express PM Control Register[8] (PME enable) is set. |
| PCI Express Slot Status Register[1] (power fault detected) is set | PCI Express Slot Control Register[1] (power fault detected enable) is set<br>and<br>PCI Express Slot Control Register[5] (hot plug interrupt enable) is set<br>and<br>either PCI Express PM Control Register[1–0] = 00 (the function power state is D0) or PCI Express PM Control Register[8] (PME enable) is set. |
| PCI Express Slot Status Register[2] (MRL sensor changed) is set | PCI Express Slot Control Register[2] (MRL sensor changed enable) is set<br>and<br>PCI Express Slot Control Register[5] (hot plug interrupt enable) is set<br>and<br>either PCI Express PM Control Register[1–0] = 00 (the function power state is D0) or PCI Express PM Control Register[8] (PME enable) is set. |
| PCI Express Slot Status Register[3] (presence detect changed) is set | PCI Express Slot Control Register[3] (presence detect changed enable) is set<br>and<br>PCI Express Slot Control Register[5] (hot plug interrupt enable) is set<br>and<br>either PCI Express PM Control Register[1–0] = 00 (the function power state is D0) or PCI Express PM Control Register[8] (PME enable) is set. |
| PCI Express Slot Status Register[4] (command completed) is set | PCI Express Slot Control Register[4] (command completed interrupt enable) is set<br>and<br>PCI Express Slot Control Register[5] (hot plug interrupt enable) is set. |

### 17.4.1.10.3 Error Conditions

Table 17-126 describes specific error types and the action taken for various transaction types.

**Table 17-126. Error Conditions**

| Transaction Type | Error Type | Action |
|---|---|---|
| Inbound response | PEX response time out. This case happens when the internal platform sends a non-posted request that did not get a response back after a specific amount of time specified in the outbound completion timeout register (PEX_OTB_CPL_TOR) | Log error (PEX_ERR_DR[PCT]) and send interrupt to PIC, if enabled. |
| Inbound response | Unexpected PEX response. This can happen if, after the response times out and the internal queue entry is deallocated, the response comes back. | Log unexpected completion error (PCI Express Uncorrectable Status Register[16]) and send interrupt to PIC, if enabled. |
| Inbound response | Unsupported request (UR) response status | Depending upon whether the initial internal request was broken up, the error is not sent until all responses come back for all portions of the internal request.<br>Log the error (PEX_ERR_DR[CDNSC] and PCI Express Uncorrectable Status Register[20]) and send interrupt to PIC, if enabled. |
| Inbound response | Completer abort (CA) response status | Depending upon whether the initial internal request was broken up, the error is not sent until all responses come back for all portions of the internal request.<br>Log the error (PEX_ERR_DR[PCAC, CDNSC] and PCI Express Uncorrectable Status Register[15] and send interrupt to PIC, if enabled. |
| Inbound response | Poisoned TLP (EP=1) | Depending upon whether the initial internal request was broken up, the error is not sent until all responses come back for all portions of the internal request.<br>Log the error (PCI Express Uncorrectable Status Register[12]) and send interrupt to PIC, if enabled. |
| Inbound response | ECRC error | Depending upon whether the initial internal request was broken up, the error is not sent until all responses come back for all portions of the internal request.<br>Log the error (PCI Express Uncorrectable Status Register[19]) and send interrupt to PIC, if enabled. |
| Inbound response | Configuration Request Retry Status (CRS) timeout for a configuration transaction that originates from PEX_CONFIG_ADDR/ PEX_CONFIG_DATA | 1.The controller always retries the transaction as soon as possible until a status other than CRS is returned. However, if a CRS status is returned after the configuration retry timeout (PEXCONF_RTY_TOR) timer expires, then the controller aborts the transaction and sends all 1s (0xFFFF_FFFF) data back to requester.<br>2. Log the error (PEX_ERR_DR[PCT]) and send interrupt to the PIC, if enabled. |
| Inbound response | UR response for configuration transaction that originates from PEX_CONFIG_ADDR/ PEX_CONFIG_DATA | 1. Send back all 1s (0xFFFF_FFFF) data.<br>2. Log the error (PEX_ERR_DR[CDNSC] and PCI Express Uncorrectable Status Register[20]) and send interrupt to PIC, if enabled. |
| Inbound response | CA response for Configuration transaction that originates from PEX_CONFIG_ADDR/ PEX_CONFIG_DATA | 1. Send back all 1s (0xFFFF_FFFF) data.<br>2. Log the error (PEX_ERR_DR[PCAC, CDNSC] and PCI Express Uncorrectable Status Register[15]) and send interrupt to PIC, if enabled. |

### Table 17-126. Error Conditions (continued)

| Transaction Type | Error Type | Action |
|---|---|---|
| Inbound response | Poisoned TLP (EP=1) response for Configuration transaction that originates from PEX_CONFIG_ADDR/ PEX_CONFIG_DATA | 1. Send back all 1s (0xFFFF_FFFF) data.<br>2. Log the error (PCI Express Uncorrectable Status Register[12]) and send interrupt to PIC, if enabled. |
| Inbound response | ECRC error response for Configuration transaction that originates from PEX_CONFIG_ADDR/ PEX_CONFIG_DATA | 1. Send back all 1s (0xFFFF_FFFF) data.<br>2. Log the error (PCI Express Uncorrectable Status Register[19]) and send interrupt to PIC, if enabled. |
| Inbound response | Configuration Request Retry Status (CRS) response for Configuration transaction that originates from ATMU | 1.The controller always retries the transaction as soon as possible until a status other than CRS is returned. However, if a CRS status is returned after the configuration retry timeout (PEXCONF_RTY_TOR) timer expires, then the controller aborts the transaction.<br>2. Log the error (PEX_ERR_DR[CRST]) and send interrupt to the PIC, if enabled. |
| Inbound response | UR response for Configuration transaction that originates from ATMU | Log the error (PEX_ERR_DR[CDNSC] and PCI Express Uncorrectable Status Register[20]) and send interrupt to PIC, if enabled. |
| Inbound response | CA response for Configuration transaction that originates from ATMU | Log the error (PEX_ERR_DR[PCAC, CDNSC] and PCI Express Uncorrectable Status Register[15]) and send interrupt to PIC, if enabled. |
| Inbound response | Malformed TLP response | PCI Express controller does not pass the response back to the core. Therefore, a completion timeout error eventually occurs. |
| Inbound request | Poisoned TLP (EP=1) | 1. If it is a posted transaction, the controller drops it.<br>2. If it is a non-posted transaction, the controller returns a completion with UR status.<br>3. Release the proper credits |
| Inbound request | ECRC error | 1. If it is a posted transaction, the controller drops it.<br>2. If it is a non-posted transaction, the controller returns a completion with UR status.<br>3. Release the proper credits |
| Inbound request | PCI Express nullified request | The packet is dropped. |
| Outbound request | Outbound ATMU crossing | Log the error (PEX_ERR_DR[OAC]). The transaction is not sent out on the link. |
| Outbound request | Illegal message size | Log the error (PEX_ERR_DR[MIS]). The transaction is not sent out on the link. |
| Outbound request | Illegal I/O size | Log the error (PEX_ERR_DR[IOIS]). The transaction is not sent out on the link. |
| Outbound request | Illegal I/O address | Log the error (PEX_ERR_DR[IOIA]). The transaction is not sent out on the link. |
| Outbound request | Illegal configuration size | Log the error (PEX_ERR_DR[CIS]). The transaction is not sent out on the link. |

Table 17-126. Error Conditions (continued)

| Transaction Type | Error Type | Action |
|---|---|---|
| Outbound response | Internal platform response with error (for example, an ECC error on a DDR read or the transaction maps to unknown address space). | Send poisoned TLP (EP=1) completion(s) for data that are known bad. If the poison data happens in the middle of the packet, the rest of the response packet(s) is also poisoned. |

## 17.4.2 Interrupts

Both INTx and message signaled interrupts (MSI) are supported; however there are differences depending on whether the PCI Express controller is configured as an RC or EP device.

### 17.4.2.1 EP Interrupt Generation

#### 17.4.2.1.1 Hardware INTx Message Generation

Hardware INTx message generation is not supported in EP mode.

#### 17.4.2.1.2 Hardware MSI Generation

In EP mode, the PCI Express controller can be configured to automatically generate MSI transactions to the root complex when an interrupt event occurs. The PCI Express controller uses *irq_out* (an internal version of the IRQ_OUT signal) to trigger the generation of the MSI. To trigger the MSI, interrupt events must be routed to *irq_out* by setting the EP (external pin) bit in the associated Interrupt Destination register in the PIC. Note that the IRQ_OUT signal should not be used for any other function if it is being used to trigger MSI transactions.

The remote root complex is expected set up the MSI capability structure of all endpoints at system initialization by filling the Message Address and Message Data registers with appropriate values and setting the MSIE bit in the MSI Message Control register.

With the PCI Express controller properly configured, when it detects the leading edge of *irq_out* going active, it generates a PCI Express memory write transaction to the address specified in the MSI Message Address register (and MSI Message Upper Address register) with a data payload as specified in the MSI Message Data register (with leading zeros appended).

#### 17.4.2.1.3 Software INTx Message Generation

Software can generate outbound assert or deassert INTx message transactions by using the outbound ATMU mechanism described in Section 17.4.1.9.1, "Outbound ATMU Message Generation."

#### 17.4.2.1.4 Software MSI Generation

Host software has to set up the MSI capability registers to enable MSI mode, and have the correct values for the MSI address and data register. Then local software has to read the MSI address in the MSI capability register and configure the outbound ATMU window to map the translated address to the MSI address. Software has to determine the number of allocated messages in the MSI capability register and allocates the appropriate data values to use. A write to the ATMU window containing the MSI address with the appropriate data value generates the desired MSI transaction to the remote RC.

### 17.4.2.2 RC Handling of INTx Message and MSI Interrupts

#### 17.4.2.2.1 INTx Message Handling

MSIs are the preferred interrupt signaling mechanism for PCI Express. However, in RC mode, the PCI Express controller supports the INTx virtual-wire interrupt signaling mechanism (as described in the PCI Express specification). Whenever the controller receives an inbound INTx (INTA, INTB, INTC, or INTD) asserted or negated message, it asserts or negates an equivalent internal INTx signal (*inta*, *intb*, *intc*, or *intd*) to the PIC.

The internal INTx signals from the PCI Express controller are logically combined with the interrupt request (IRQ*n*) input signals so that they share the same interrupt controlled by the associated EIVPR*n* and EIDR*n* registers in the PIC. Refer to Chapter 9, “Programmable Interrupt Controller (PIC),” for more information about handling of PCI Express INTx interrupts and the external interrupt request (IRQ*n*) signals.

If a PCI Express INTx interrupt is being used, then the PIC must be configured so that external interrupts are active-low (EIVPRn[P] = 0), and level-sensitive (EIVPRn[S] = 1).

#### 17.4.2.2.2 MSI Handling

An inbound MSI cycle must hit into the PEXCSRBAR window with the address offset that points to the MSIIR register in the PIC. Note that it is the responsibility of the host software to configure each EP’s MSI capability registers such that an MSI cycle generated from the EP device is routed to the MSIIR register in the PIC and for the appropriate interrupt to be generated to the core.

## 17.4.3 Initial Credit Advertisement

To prevent overflowing of the receiver’s buffers and for ordering compliance purposes, the transmitter cannot send transactions unless it has enough flow control (FC) credits to send. Each device maintains an FC credit pool. The FC information is conveyed between the two link partners by DLLPs during link training (initial credit advertisement). The transaction layer performs the FC accounting functions. One FC unit is four DWs (16-bytes) of data.

**Table 17-127. Initial credit advertisement**

| Credit Type | Initial Credit Advertisement |
|---|---|
| PH (Memory Write, Message Write) | 4 |
| PD (Memory Write, Message Write) | (256/16)x4=64 |
| NPH (Memory Read, IO Read, Cfg Read, Cfg Write) | 8 |
| NPD (IO Write, Cfg Write) | 2 |

Table 17-127. Initial credit advertisement

| Credit Type | Initial Credit Advertisement |
|---|---|
| CPLH (Memory Read Completion, IO R/W Completion, Cfg R/W Completion) | Infinite |
| CPLD (Memory Read Completion, IO Read Completion, Cfg Read Completion) | Infinite |

## 17.4.4 Power Management

All device power states are supported with the exception of D3cold with Vaux. In addition, all link power states are supported with the exception of L2 states. Only L0s ASPM mode is supported if enabled by configuring the Link Control register’s bits 1–0 in configuration space. Note that there is no power saving in the controller when the device is put into a non-D0 state. The only power saving is the I/O drivers when the controller is put into a non-L0 link state.

Table 17-128. Power Management State Supported

| Component D-State | Permissible Interconnect State | Action |
|---|---|---|
| D0 | L0, L0s | In full operation. |
| D1 | L0, L0s, L1 | All outbound traffics are stalled. All inbound traffic is thrown away. The only exceptions are PME messages and configuration transactions. If the device is in RC mode, it is permissible to send a PM_Turn_Off message through the PEX Power Management Command register. |
| D2 | L0, L0s, L1 | All outbound traffics are stalled. All inbound traffic is thrown away. The only exceptions are PME messages and configuration transactions. If the device is in RC mode, it is permissible to send a PM_Turn_Off message through the PEX Power Management Command register. |
| D3hot | L0, L0s, L1, L2/L3 Ready | All outbound traffics are stalled. All inbound traffic is thrown away. The only exceptions are PME messages and configuration transactions. If the device is in RC mode, it is permissible to send a PM_Turn_Off message through the PEX Power Management Command register. Note that if a transition of D3hot->D0 occurs, a reset is performed to the controller’s configuration space. In addition, link training restarts. |
| D3cold | L3 | Completely off. |

### 17.4.4.1 L2/L3 Ready Link State

The L2/L3 Ready link state is entered after the EP device is put into a D3hot state followed by a PME_Turn_Off/PME_TO_Ack message handshake protocol. Exiting this state requires a POR reset or a $\overline{\text{WAKE}}$ signal from the EP device. The PCI Express controller (in EP mode) does not support the generation of beacon; therefore, as an alternative, the device can use one of the GPIO signals as an enable to an external tristate buffer to generate a $\overline{\text{WAKE}}$ signal, as shown in Figure 17-132.

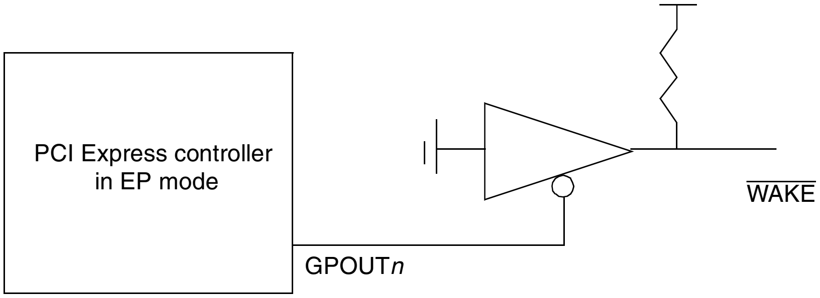


**Figure 17-132. $\overline{\text{WAKE}}$ Generation Example**

In RC mode, the $\overline{\text{WAKE}}$ signal from the EP device can be connected to one of the external interrupt inputs to service the $\overline{\text{WAKE}}$ request.

### 17.4.5 Hot Reset

When a hot reset condition occurs, the controller (in both RC and EP mode) initiates a clean-up of all outstanding transactions and returns to an idle state. All configuration register bits that are non-sticky are reset. Link training takes place subsequently. The device is permitted to generate a hot reset condition on the bus when it is configured as an RC device by setting the “Secondary Bus Reset” bit in the Bridge Control Register in the configuration space. As an EP device, it is not permitted to generate a hot reset condition; it can only detected a hot reset condition and initiates the clean-up procedure appropriately.

### 17.4.6 Link Down

Typically, a link down condition occurs after a hot reset event; however, it is possible for the link to go down unexpectedly without a hot reset event. When this occurs, a link down condition is detected (PEX_PME_MSG_DR[LDD]=1). Link down is treated similarly to a hot reset condition.

Subsequently, while the link is down, all new posted outbound transactions are discarded. All new non-posted ATMU transactions are errored out. Non-posted configuration transactions issued using PEX_CONFIG_ADDR/PEX_CONFIG_DATA toward the link returns 0xFFFF_FFFF (all 1s). As soon as the link is up again, the sending of transaction resumes.

Note that in EP mode, a link down condition causes the controller to reset all non-sticky bits in its PCI Express configuration registers as if it had been hot reset.

## 17.5 Initialization/Application Information

### 17.5.1 Boot Mode and Inbound Configuration Transactions

In normal boot mode (cfg_cpu_boot = 1), the core is allowed to boot and configure the device. During this time, the PCI Express interface retries all inbound PCI Express configuration transactions. When the core has configured the device to a state where it can accept inbound PCI Express configuration transactions, the boot code should set the CFG_READY bit in the PEX_CFG_READY register after which inbound

PCI Express configuration transactions are accepted. Refer to Section 17.3.10.18, “Configuration Ready Register—0x4B0,” for more information about the CFG_READY bit.

In boot hold-off mode (cfg_cpu_boot = 0), the core is prevented from fetching its first instruction by withholding its internal bus grant. During this time, the PCI Express interface accepts all inbound PCI Express configuration transactions which allows an external host/RC to configure the device. When the external host/RC has configured the device to a state where it can allow the core to fetch code from the boot vector, it sets the EEBPCR[CPU_EN] bit after which the core is granted the internal bus.

# Chapter 18
# Enhanced Serial Peripheral Interface

The enhanced serial peripheral interface (eSPI) allows the device to exchange data with peripheral devices such as EEPROMs, real-time clocks, A/D converters, and ISDN devices. The eSPI is a full-duplex, synchronous, character-oriented channel that supports a simple interface (receive, transmit, clock and chip selects). The eSPI consists of transmitter and receiver sections, an independent baud-rate generator, and a control unit. The transmitter and receiver sections use the same clock, which is derived from the eSPI baud rate generator in master mode. During an eSPI transfer, data is sent and received simultaneously.

The eSPI receiver and transmitter each have a FIFO of 32 bytes, as shown in Figure 18-1. When the eSPI is disabled in the eSPI mode register (SPMODE[EN] = 0), it consumes little power.

## 18.1 Introduction

The eSPI block diagram is shown in Figure 18-1.

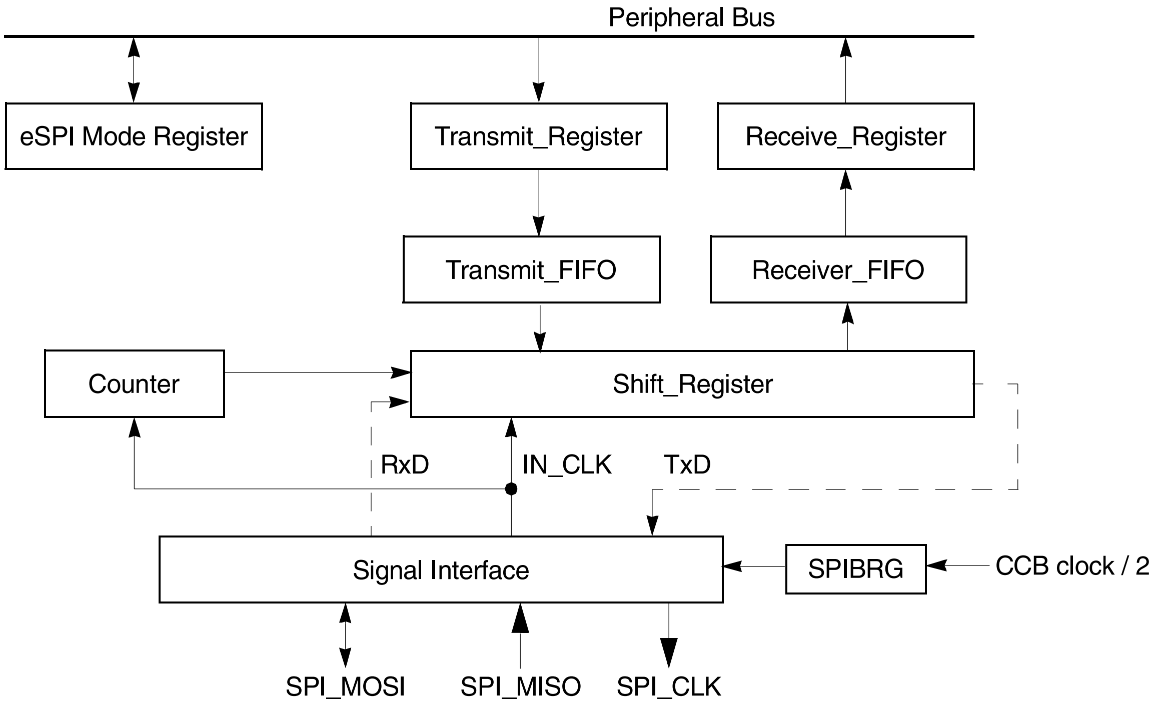


**Figure 18-1. eSPI Block Diagram**

### 18.1.1 Features

The major features of the eSPI are listed as follows:

- Interface contains SPI_MOSI, SPI_MISO, SPI_CLK, and 4 chip selects
- Supports eSPI master
- Supports RapidS$^{TM}$ full clock cycle operation
- Full-duplex or half-duplex master operation
- Supports Winbond dual output read
- Command in transaction level—easier for accessing eSPI devices
- Works with a range from 4-bit to 16-bit data characters
- Supports back-to-back character transmission and reception
- supports reverse data mode for 8/16 bits data characters
- Supports single-master environment
- Maximum clock rate possible is (system clock rate/4)
- Independent programmable baud rate generator
- Programmable clock phase and polarity.
- Supports 4 different configurations per chip select
- Local loopback capability for testing
- Supports booting from eSPI interface. See Section 4.5.1.2, "eSPI Boot ROM," for more detailed information.

### 18.1.2 eSPI Transmission and Reception process

As the eSPI is a character-oriented communication unit, the core is responsible for packing and unpacking the receive and transmit frames. A frame consists of all of the characters transmitted or received during a completed eSPI transmission session, from the first character written to the SPITF register to the last character transmitted with the total number as indicated in the command written to the SPCOM register. See Section 18.3.1.4, "eSPI Command Register (SPCOM)," for more information.

The core receives data by reading the eSPI receive data register (SPIRF) when the NE ("not empty") bit in the eSPI event register (SPIE) is set.

The core transmits data by writing it into the SPITF. After the core writes the final character to SPITF it waits for DON bit in the SPIE to be set indicating frame was fully transmitted. It might then write a new command to SPCOM.

The eSPI sets the NF ("not full") bit in SPIE whenever its transmit FIFO is not full.

The eSPI core handshake protocol can be implemented by using a polling or interrupt mechanism. When using a polling mechanism, the core reads the SPIE in a predefined frequency and acts according to the value of the SPIE bits. The polling frequency depends on the eSPI serial channel frequency. When using the interrupt mechanism, setting either the TNF (not full) or RNE (not empty) bits of the SPIE causes an interrupt to the core. The core then reads the SPIE and acts appropriately.

### 18.1.3 Modes of Operation

The eSPI can be programmed to work in a single master environment. This section describes eSPI master operation in a single-master configuration.

In master mode, the eSPI sends a message to the slave peripheral, which sends back a simultaneous reply. A single master with multiple slaves uses up to 4 chip select signals to selectively enable slaves, as shown in Figure 18-2.

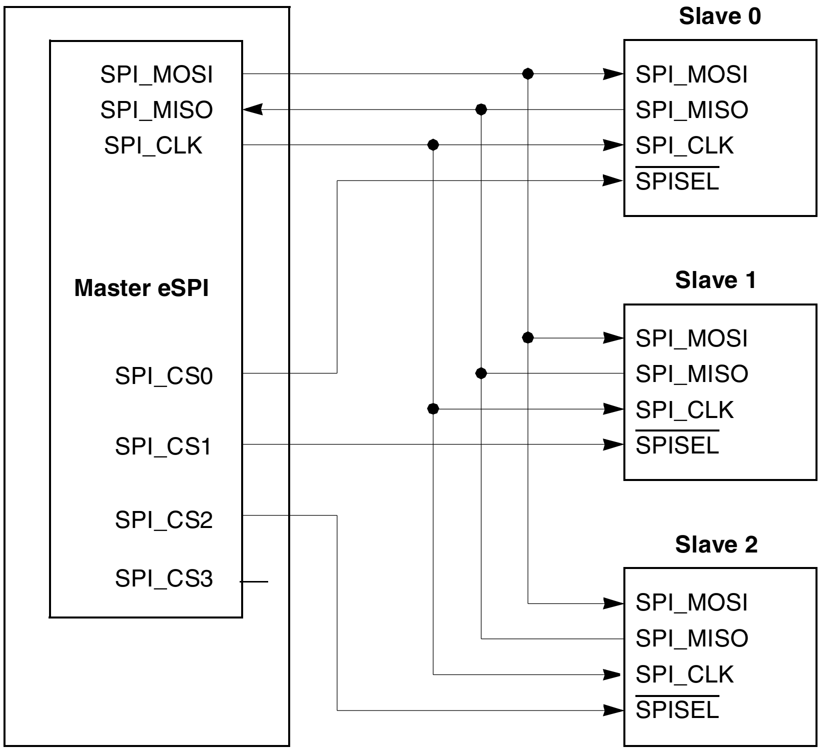


**Figure 18-2. Single-Master/Multi-Slave Configuration**

To start exchanging data, the core writes the data to be sent into the SPITF register. The eSPI then generates programmable clock pulses on SPI_CLK for each character. It shifts Tx data out on the “eSPI master-out slave-in” (SPI_MOSI) and Rx data in on the eSPI “master-in slave-out” (SPI_MISO) simultaneously. During the transmission process the core is responsible for supplying the data whenever the eSPI requests it to ensure smooth operation.

The maximum sustained data rate that the eSPI supports depends on the SW latency. However, the eSPI can transfer a single character at very high rates— system clock/2 up to a maximum specified by the device hardware specifications. Gaps might be inserted between multiple frames.

## 18.2 External Signal Descriptions

The eSPI interface consists of transmit, receive, clock and chip selects

## 18.2.1 Overview

Table 18-1 lists signal properties.

**Table 18-1. Signal Properties**

| Name | Function |
|---|---|
| SPI_MISO | master input slave output |
| SPI_MOSI | master output slave input or second master input slave output for Winbond dual output read |
| SPI_CLK | ioutput serial clock connected to the other SPI_CLK |
| $\overline{\text{SPI\_CS}}$[0:3] | eSPI slave select outputs |

## 18.2.2 Detailed Signal Descriptions

Table 18-2 describes the signals in detail.

**Table 18-2. Detailed Signal Descriptions**

| Signal | I/O | Description | |
|---|---|---|---|
| SPI_MISO | I | master input slave output | |
| | | **State Meaning** | Asserted—the data that has been received from the eSPI is high<br>Negated—the data that has been received from the eSPI is low |
| | | **Timing** | Assertion—according to the SPI_CLK assertion/negation/in the middle of phase (depends on the SPMODEx configuration register).<br>Negation—according to the SPI_CLK assertion/negation/in the middle of phase (depends on the SPMODEx configuration register) |
| SPI_MOSI | O | master output slave input or 2nd master input slave output for Winbond dual output read | |
| | | **State Meaning** | Asserted—the data that has been transmitted from/to the eSPI is high<br>Negated—the data that has ben transmitted from/to the eSPI is low |
| | | **Timing** | Assertion—according to the SPI_CLK assertion/negation/in the middle of phase (depends on the SPMODEx configuration register).<br>Negation—according to the SPI_CLK assertion/negation/in the middle of phase (depends on the SPMODEx configuration register) |
| SPI_CLK | O | Serial clock out | |
| | | **State Meaning** | Assertion/Negation according to SPMODEx[PM,DIV16] register rate configuration |
| | | **Timing** | Assertion/Negation—during frame reception/transmission |
| $\overline{\text{SPI\_CS}}$[0:3] | O | eSPI slave select outputs | |
| | | **State Meaning** | Asserted— slave 0, 1, 2, 3 is selected and master controls transmission/reception<br>Negated—idle state |
| | | **Timing** | Assertion—a predefined time before frame starts, during frame transmission/reception, a predefined time after frame ends<br>Negation—when master is idle or controls another slave |

The eSPI can be configured as a master in single master environment. The master eSPI generates the transfer clock SPI_CLK using the eSPI baud rate generator (BRG). The eSPI BRG takes as its input the platform clock divided by two.

SPI_CLK is a gated clock, active only during data transfers. Four combinations of SPI_CLK phase and polarity can be configured with the clock invert SPMODEx[CIx] and clock phase SPMODEx[CPx] register bits.

The eSPI master-in slave-out SPI_MISO signal acts as an input for master devices and as an output for slave devices. Conversely, the master-out slave-in SPI_MOSI signal is an output for master devices and an input for slave devices. However, it also acts as a second input for master devices and as a second output for slave devices when using Winbond dual output read.

SPI_CLK is the clock output signal that shifts received data in from SPI_MISO and transmitted data out to SPI_MOSI. eSPI masters must output a slave select signal to enable eSPI slave devices.

## 18.3 Memory Map/Register Definition

Table 18-3 shows the memory mapped registers of the eSPI and their offsets. It lists the offset, name, and a cross-reference to the complete description of each register. Note that the full register address is comprised of CCSRBAR together with the SPI block base address and offset listed in Table 18-3. Undefined 4-byte address spaces within offset 0x000–0xFFF are reserved.

**Table 18-3. eSPI Registers**

<table>
<tr><th colspan="5">Enhanced Serial Peripheral Interface (eSPI)—Block Base Address 0x0_7000</th></tr>
<tr><th>Offset</th><th>Register</th><th>Access</th><th>Reset Value</th><th>Section/Page</th></tr>
<tr><td>0x000</td><td>SPMODE—eSPI mode register</td><td>R/W</td><td>0x0000_100F</td><td>18.3.1.1/18-6</td></tr>
<tr><td>0x004</td><td>SPIE—eSPI event register</td><td>Mixed</td><td>0x0020_0000</td><td>18.3.1.2/18-6</td></tr>
<tr><td>0x008</td><td>SPIM—eSPI mask register</td><td>R/W</td><td>0x0000_0000</td><td>18.3.1.3/18-7</td></tr>
<tr><td>0x00C</td><td>SPCOM—eSPI command register</td><td>W</td><td>0x0000_0000</td><td>18.3.1.4/18-9</td></tr>
<tr><td>0x010</td><td>SPITF—eSPI transmit FIFO access register</td><td>W</td><td>—</td><td>18.3.1.5/18-10</td></tr>
<tr><td>0x014</td><td>SPIRF—eSPI receive FIFO access register</td><td>R</td><td>—</td><td>18.3.1.6/18-11</td></tr>
<tr><td>0x018–0x01C</td><td>Reserved</td><td colspan="3">—</td></tr>
<tr><td>0x020</td><td>SPMODE0—eSPI CS0 mode register</td><td>R/W</td><td>0x0010_0000</td><td>18.3.1.7/18-12</td></tr>
<tr><td>0x024</td><td>SPMODE1—eSPI CS1 mode register</td><td>R/W</td><td>0x0010_0000</td><td>18.3.1.7/18-12</td></tr>
<tr><td>0x028</td><td>SPMODE2—eSPI CS2 mode register</td><td>R/W</td><td>0x0010_0000</td><td>18.3.1.7/18-12</td></tr>
<tr><td>0x02C</td><td>SPMODE3—eSPI CS3 mode register</td><td>R/W</td><td>0x0010_0000</td><td>18.3.1.7/18-12</td></tr>
</table>

## 18.3.1 Register Descriptions

### 18.3.1.1 eSPI Mode Register (SPMODE)

The eSPI mode register (SPMODE), shown in Figure 18-3, controls eSPI general operation mode.

Offset 0x000 Access: Read/Write

| | 0 | 1 | 2–17 | 18–23 | 24–26 | 27–31 |
|---|---|---|---|---|---|---|
| R<br>W | EN | LOOP | — | TXTHR | — | RXTHR |
| Reset | 0 | 0 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 | 0 0 0 | 0 1 1 1 1 |

**Figure 18-3. eSPI Mode Register (SPMODE)**

Table 18-4 describes the SPMODE fields.

**Table 18-4. SPMODE Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | EN | Enable eSPI. Any bits in SPMODE must not change when EN is set.<br>0 The eSPI is disabled. The eSPI is in a idle state and consumes minimal power. The eSPI BRG is not functioning and the input clock is disabled.<br>1 The eSPI is enabled. |
| 1 | LOOP | Loop mode. Enables local loopback operation.<br>0 Normal operation.<br>1 Loopback mode. Used to test the eSPI controller internal functionality, the transmitter output is internally connected to the receiver input. The receiver and transmitter operate normally, except that received data is ignored. |
| 2–17 | — | Reserved |
| 18–23 | TXTHR | Tx FIFO threshold—if Tx FIFO has less than TXTHR bytes, an interrupt can be issued to the core.<br>Valid values: 1–32 |
| 27–31 | RXTHR | Rx FIFO threshold—if Rx FIFO has more than RXTHR bytes, an interrupt can be issued to the core.<br>Valid values: 0–31 |

### 18.3.1.2 eSPI Event Register (SPIE)

The eSPI event register (SPIE) generates interrupts and reports events recognized by the eSPI. When an event is recognized, the eSPI sets the corresponding SPIE bit. Clear SPIE bits by writing a 1—writing 0 has no effect. Setting a bit in the eSPI mask register (SPIM) enables and clearing a bit masks the corresponding interrupt. Unmasked SPIE bits must be cleared before the core clears internal interrupt requests. Bits RNE and TNF are status bits. Fields RXCNT and TXCNT hold Rx and Tx fifos statuses. They are not cleared as a result of writing to SPIE.

Figure 18-4 shows the eSPI event register.

Offset 0x004 Access: Mixed

| | 0 | 1 | 2 | | | | | 7 | 8 | 9 | 10 | | | | | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | — | | RXCNT | | | | | | — | | TXCNT | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

| | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | | | | | | | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | TXE | DON | RXT | RXF | TXT | — | RNE | TNF | — | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | All zeros | | | | | | | | | | | | | | | |

**Figure 18-4. eSPI Event Register (SPIE)**

Table 18-5 describes the SPIE fields.

**Table 18-5. SPIE Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–1 | — | Reserved, should be cleared. |
| 2–7 | RXCNT | The current number of full Rx FIFO bytes<br>**Note**—For character lengths of 9 to 16 bits—each character occupies 2 bytes in Rx/Tx FIFO |
| 8–9 | — | Reserved, should be cleared. |
| 10–15 | TXCNT | The current number of free Tx FIFO bytes<br>**Note**—For character lengths of 9 to 16 bits—each character occupies 2 bytes in Rx/Tx FIFO |
| 16 | TXE | Tx FIFO is empty |
| 17 | DON | Last character was transmitted .<br>The last character was transmitted and a new command can be written for the next frame |
| 18 | RXT | Rx FIFO has more than RXTHR bytes i.e. at least RXTHR+1 bytes |
| 19 | RXF | Rx FIFO is full |
| 20 | TXT | Tx FIFO has less than TXTHR bytes.i.e. at most TXTHR–1 bytes |
| 21 | — | Reserved, should be cleared. |
| 22 | RNE | Not empty. Indicates that the Rx FIFO register contains a received character.<br>0 The Rx FIFO is empty<br>1 The Rx FIFO has a received character. The core can read the content of Rx FIFO through SPIRF. |
| 23 | TNF | Tx FIFO not full.<br>0 The transmitter FIFO is full.<br>1 The transmitter FIFO is not full. |
| 24–31 | — | Reserved, should be cleared. |

### 18.3.1.3 eSPI Mask Register (SPIM)

The eSPI mask register (SPIM) enables/masks interrupts for events recognized by the eSPI. When an event is recognized, the eSPI sets the corresponding SPIE bit. Setting a bit in the eSPI mask register (SPIM) enables and clearing a bit masks the corresponding interrupt. Unmasked SPIE bits must be cleared before the core clears internal interrupt requests.

Bits RNE and TNF in SPIE are status bits. They are not cleared as a result of writing to SPIE. Figure 18-5 shows the eSPI mask register.

Offset 0x008 Access: Read/Write

| | 0–15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24–31 |
|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | TXE | DON | RXT | RXF | TXT | — | RNE | TNF | — |
| Reset | All zeros | | | | | | | | | |

**Figure 18-5. eSPI Mask Register (SPIM)**

Table 18-6 describes the SPIM fields.

**Table 18-6. SPIM Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–15 | — | Reserved, should be cleared. |
| 16 | TXE | Tx FIFO empty interrupt mask<br>0 TXE event will not cause eSPI Interrupt<br>1 TXE event will cause eSPI Interrupt |
| 17 | DON | Last character transmitted mask<br>0 DON event will not cause eSPI Interrupt<br>1 DON event will cause eSPI Interrupt |
| 18 | RXT | Rx threshold interrupt mask<br>0 RXT event will not cause eSPI Interrupt<br>1 RXT event will cause eSPI Interrupt |
| 19 | RXF | Rx FIFO full interrupt mask<br>0 RXF event will not cause eSPI Interrupt<br>1 RXF event will cause eSPI Interrupt |
| 20 | TXT | Tx threshold interrupt mask<br>0 TXT event will not cause eSPI Interrupt<br>1 TXT event will cause eSPI Interrupt |
| 21 | — | Reserved, should be cleared. |
| 22 | RNE | Rx not empty interrupt mask<br>0 Not Empty event will not cause eSPI Interrupt<br>1 Not Empty event will cause eSPI Interrupt |
| 23 | TNF | Tx not full interrupt mask<br>0 Not full event will not cause eSPI Interrupt<br>1 Not full event will cause eSPI Interrupt |
| 24–31 | — | Reserved, should be cleared. |

### 18.3.1.4 eSPI Command Register (SPCOM)

The eSPI command register (SPCOM), shown in Figure 18-6, is used by the host to supply information on the new frame.

After SPCOM has been written to initiate the first transaction after startup, commands can be executed only after SPIE[DON] is set. Otherwise they are ignored.

Offset 0x00C

Access: Write only

| | 0 1 | 2 | 3 | 4 | 5 | 6 7 | 8 15 | 16 31 |
|---|---|---|---|---|---|---|---|---|
| R | | | | | | — | | |
| W | CS | RxDelay | DO | TO | HLD | | RxSKIP | TRANLEN |
| Reset | All zeros | | | | | | | |

**Figure 18-6. eSPI Command Register (SPCOM)**

Table 18-7 describes the SPCOM fields.

**Table 18-7. SPCOM Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–1 | CS | Chip select—chip select for which transaction is destined<br>00 $\overline{\text{SPI\_CS0}}$<br>01 $\overline{\text{SPI\_CS1}}$<br>10 $\overline{\text{SPI\_CS2}}$<br>11 $\overline{\text{SPI\_CS3}}$ |
| 2 | RxDelay | 0 Normal eSPI operation<br>1 Rx data should be sampled a bit later than regular eSPI (used for full cycle operation such as with Atmel RapidS devices) |
| 3 | DO | 0 Normal eSPI operation<br>1 Winbond dual output read—when eSPI master reads data 2 data bits are available (on MISO and MOSI)<br>This mode is usefull only for character lengths of 4,6,8 .<br>DO and RapidS should not be set simultaneously. |
| 4 | TO | Transmit only<br>1 No reception is done for the frame (usefull for write transactions)<br>0 Normal operation |
| 5 | HLD | 0 Normal operation<br>1 Mask first generated SPI_CLK. Should be used only for RapidS mode0 |
| 6–7 | — | Reserved, should be cleared. |
| 8–15 | RxSKIP | If (RXSKIP != 0)—Number of characters skipped for reception from frame start.<br>Non-zero values of RxSKIP force the eSPI to half-duplex mode, and therefore this causes TRANLEN–RxSKIP characters to be skipped for transmission.<br>RXSKIP is useful for reads of SPI Flash memories where the first valid read data is received several characters after the transmission begins (after the eSPI has transmitted an instruction opcode and address).<br>**Note:** If TO=1 RxSKIP is ignored.<br>If RXSKIP=0 and TO=0, the eSPI changes to full duplex mode. |
| 16–31 | TRANLEN | Transaction length – (number of characters in the frame – 1) |

A transaction can be full duplex (regular eSPI) or half duplex. Half duplex can be used for example for write accesses to a flash (only transmit) or for a read access from a flash (first part is transmit without receive, while the second part is receive without transmit).

### 18.3.1.5 eSPI Transmit FIFO Access Register (SPITF)

The 32-bit write-only eSPI transmit FIFO access register (SPITF) holds the characters to be written to the transmit FIFO. The number of bits in each character is specified by SPMODEx[LENx]. Each time SPIE[TNF] is set, the core can write more data to the SPITF register, if there is no error indication in the SPIE.

For character lengths of 4 to 8 bits, SPITF contains up to 4 characters (unless end of frame). The lsbs are in bits 7, 15, 23, and 31 of SPITF.

For character lengths of 9 to 16 bits, SPITF contains up to 2 characters (unless end of frame). For 16 bits with SPMODEx[REVx]=1, the lsb is in bits 15 and 31 of SPITF. For other options, lsbs are in bits 7 and 23 while msbs are in bits (23–LENx) and (39–LENx) of SPITF.

Example : REV=0, LEN=10 (0xA), SPITF[0–15] = 0xFB05—bitstream is : (lsb first) 11011111101 (msb last).

**Note**—The user must write N bytes of SPITF (1<=N<=4) that do not exceed the number of free bytes in the transmit FIFO. It is valid for the user to write only 1 or 2 bytes of SPITF (at offset 0x010) if the user wishes to write fewer characters than the maximum supported by SPITF for the particular character length in use.

Figure 18-7 shows the eSPI transmit data register.

Offset 0x010 Access: Write only

| | 0 … 31 |
|---|---|
| R | |
| W | DATA |
| Reset | All zeros |

**Figure 18-7. eSPI Transmit Data Register (SPITF)**

The following figures show examples of the contents of SPITF with various parameters set.

| 0 1 2 3 | 4 | 5 6 | 7 | 8 9 10 11 | 12 | 13 14 | 15 | 16 17 18 19 | 20 | 21 22 | 23 | 24 25 26 27 | 28 | 29 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| — | MSB0 | Data 0 | LSB0 | — | MSB1 | Data 1 | LSB1 | — | MSB2 | Data 2 | LSB2 | — | MSB3 | Data 3 | LSB3 |

**Figure 18-8. SPITF Example—SPMODEx[REVx]=0, SPMODEx[LENx]=3, LSB Sent First**

| 0 | 1 2 3 4 5 6 | 7 | 8 | 9 10 11 12 13 14 | 15 | 16 | 17 18 19 20 21 22 | 23 | 24 | 25 26 27 28 29 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| MSB0 | Data 0 | LSB0 | MSB1 | Data 1 | LSB1 | MSB2 | Data 2 | LSB2 | MSB3 | Data 3 | LSB3 |

**Figure 18-9. SPITF Example—SPMODEx[REVx]=x, SPMODEx[LENx]=7**

| 0 1 2 3 4 5 6 | 7 | 8 9 10 11 12 | 13 | 14 15 | 16 17 18 19 20 21 22 | 23 | 24 25 26 27 28 | 29 | 30 31 |
|---|---|---|---|---|---|---|---|---|---|
| Data 0 LS Byte | LSB0 | — | MSB0 | Data 0 MS Byte | Data 1 LS Byte | LSB1 | — | MSB1 | Data 1 MS Byte |

**Figure 18-10. SPITF Example—SPMODEx[REVx]=0, SPMODEx[LENx]=10, LSB Sent First**

| 0 1 2 3 4 5 6 | 7 | 8 | 9 10 11 12 13 14 15 | 16 17 18 19 20 21 22 | 23 | 24 | 25 26 27 28 29 30 31 |
|---|---|---|---|---|---|---|---|
| Data 0 LS Byte | LSB0 | MSB0 | Data 0 MS Byte | Data 1 LS Byte | LSB1 | MSB1 | Data 1 MS Byte |

**Figure 18-11. SPITF Example—SPMODEx[REVx]=0, SPMODEx[LENx]=15, LSB Sent First**

| 0 | 1 2 3 4 5 6 7 | 8 9 10 11 12 13 14 | 15 | 16 | 17 18 19 20 21 22 23 | 24 25 26 27 28 29 30 | 31 |
|---|---|---|---|---|---|---|---|
| MSB0 | Data 0 MS Byte | Data 0 LS Byte | LSB0 | MSB1 | Data 1 MS Byte | Data 1 LS Byte | LSB1 |

**Figure 18-12. SPITF Example—SPMODEx[REVx]=1, SPMODEx[LENx]=15, MSB Sent First**

### 18.3.1.6 eSPI Receive FIFO Access Register (SPIRF)

The 32-bit read-only eSPI receive data register (SPIRF) is used to hold characters read from the receive FIFO. Each time SPIE[RNE] is set, the core can read the SPIRF register.

For character lengths of 4 to 8 bits, SPIRF contains up to 4 characters. The msbs are in bits 0, 8, 16, and 24. For character lengths of 9 to 16 bits, SPIRF contains up to 2 characters. The msbs are in bits 0 and 16. SPMODEx[REVx] does not affect the msb or lsb bit positions when reading the SPIRF register.

The user must read N bytes of SPIRF ($1 \leq N \leq 4$) that do not exceed the amount of data in the receive FIFO. The user can read less bytes than the amount of data in the receive FIFO. For example, a 1-byte read of SPIRF when configured for 8-bit characters with 4 characters of data in the receive FIFO results in the 3 unread characters shuffling down to the lower 24 bits of SPIRF in preparation for the following SPIRF read.

Offset 0x014 Access: Read only

| | 0 … 31 |
|---|---|
| R | DATA |
| W | |
| Reset | All zeros |

**Figure 18-13. eSPI Receive Data Register (SPIRF)**

The following tables show examples of the contents of SPIRF with various parameters set.

| 0 | 1 2 | 3 | 4 7 | 8 | 9 10 | 11 | 12 15 | 16 | 17 18 | 19 | 20 23 | 24 | 25 26 | 27 | 28 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MSB0 | Data 0 | LSB0 | — | MSB1 | Data 1 | LSB1 | — | MSB2 | Data 2 | LSB2 | — | MSB3 | Data 3 | LSB3 | — |

**Figure 18-14. SPIRF Example—SPMODEx[LENx]=3**

| 0 | 1 7 | 8 9 | 10 | 11 15 | 16 | 17 23 | 24 25 | 26 | 27 31 |
|---|---|---|---|---|---|---|---|---|---|
| MSB0 | Data 0 MS Byte | Data 0 LS Byte | LSB0 | — | MSB1 | Data 1 MS Byte | Data 1 LS Byte | LSB1 | — |

**Figure 18-15. SPIRF Example—SPMODEx[LENx]=10**

| 0 | 1 7 | 8 14 | 15 | 16 | 17 23 | 24 30 | 31 |
|---|---|---|---|---|---|---|---|
| MSB0 | Data 0 MS Byte | Data 0 LS Byte | LSB0 | MSB1 | Data 1 MS Byte | Data 1 LS Byte | LSB1 |

**Figure 18-16. SPIRF Example—SPMODEx[LENx]=15**

### 18.3.1.7 eSPI CS*n* Mode Registers (SPMODE0–4)

The eSPI CS*n* mode registers (SPMODE*n*), shown in Figure 18-17, control eSPI master operation with chip select *n*.

Offset SPMODE0: 0x020
SPMODE1: 0x024
SPMODE2: 0x028
SPMODE3: 0x02C

Access: Read/Write

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | CI*n* | CP*n* | REV*n* | DIV16*n* | PM*n* | | | | ODD*n* | — | | POL*n* | LEN*n* | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

| | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | CS*n*BEF | | | | CS*n*AFT | | | | CS*n*CG | | | | | — | | |
| Reset | All zeros | | | | | | | | | | | | | | | |

**Figure 18-17. eSPI CS*n* Mode Register (SPMODE*n*)**

Table 18-8 describes the SPMODE*n* fields.

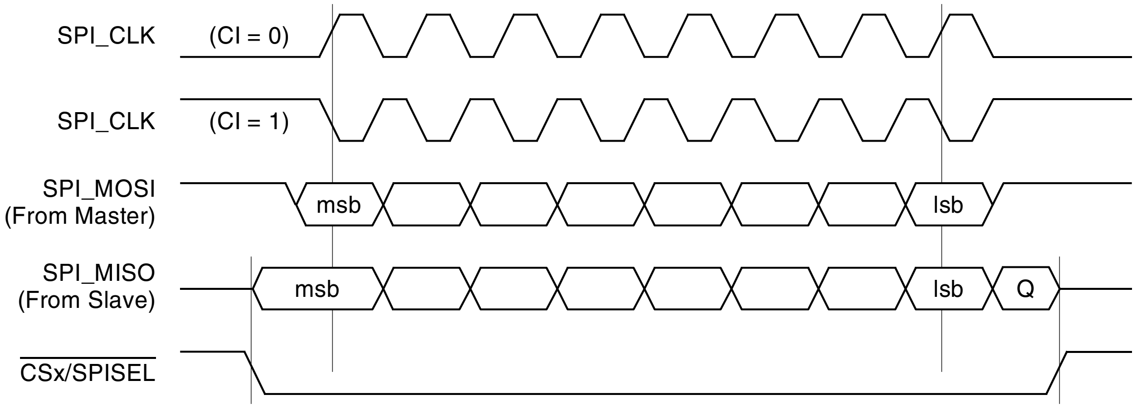
**Table 18-8. SPMODE*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | CI*n* | Clock invert. Inverts eSPI clock polarity. See Figure 18-18 and Figure 18-19 for more information<br>0 The inactive state of SPI_CLK is low.<br>1 The inactive state of SPI_CLK is high. |
| 1 | CP*n* | Clock phase. Selects the transfer format. See Figure 18-18 and Figure 18-19 for more information.<br>0 SPI_CLK starts toggling at the middle of the data transfer.<br>1 SPI_CLK starts toggling at the beginning of the data transfer. |
| 2 | REV*n* | Reverse data mode. Determines the receive and transmit character bit order.<br>0 lsb of the character sent and received first<br>1 msb of the character sent and received first - for 8/16 bits data character only |
| 3 | DIV16*n* | Divide by 16. Selects the clock source for the eSPI baud rate generator(eSPI BRG) when configured as an eSPI master.<br>0 System clock is the input to the eSPI BRG.<br>1 System clock/16 is the input to the eSPI BRG.<br>**Note:** System clock is defined to be CCB clock divided by 2 |
| 4–7 | PM*n* | Prescale modulus select. Specifies the divide ratio of the prescale divider in the eSPI clock generator. The eSPI baud rate generator clock source (either system clock or system clock divided by 16, depending on DIV16 bit) is divided by 2* ([PM] + 1), a range from 2 to 32. For example, if the prescale modulus is set to PM=0x0011 and DIV16 is set, the SPI_CLK/system clock rate will be 16*(2*(0x0011+1))=128<br>**Note:** System clock is defined to be CCB clock divided by 2 |
| 8 | ODD*n* | 0 Even division: 2*(PM+1)*(15*DIV16+1); 50% duty cycle<br>1 Odd division: (2*PM + 1)*(15*DIV16+1) (except for PM=0 where it divides by 2*(7*DIV16+1)); duty cycle is (PM+1)/(2*PM+1) for DIV16=0; duty cycle is 50% for DIV16=1 |
| 9–10 | — | Reserved, should be cleared. |

**Table 18-8. SPMODE*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 11 | POL*n* | CS*n* Polarity.<br>1 Asserted Low, Negated High<br>0 Asserted High, Negated Low. |
| 12–15 | LEN*n* | Character length in bits per character. Must be between 00011 (4 bits) and 01111 (16 bits). A value less than 4 causes erratic behavior. |
| 16–19 | CS*n*BEF | CS assertion time in bits before frame start (i.e. before clock toggles)<br>Example: CS0BEF =0010 inserts 2bits time gap between CS0 assertion to clock toggle |
| 20–23 | CS*n*AFT | CS assertion time in bits after frame end (i.e. after clock finishes toggling)<br>Example: CS0AFT =0010 inserts 2bits time gap between clock stop to CS0 negation |
| 24–28 | CS*n*CG | Clock gap<br>insert gaps between transmitted frames according to this size (during this time, chip select is negated). Chip select is negated minimum time of 1 bit time.<br>Example: CS0CG =00101 inserts 5+1=6 bits time gap between every two consecutive frames |
| 29–31 | — | Reserved, should be cleared. |

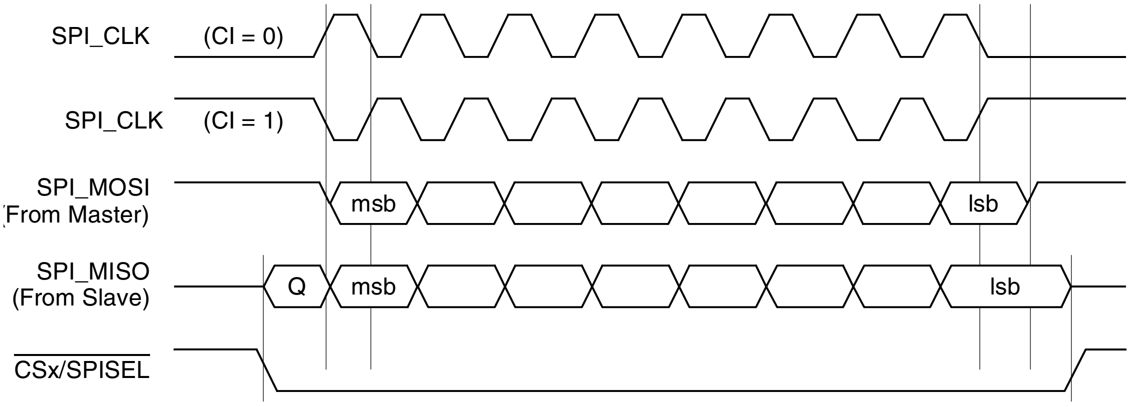
Figure 18-18 shows the eSPI transfer format in which SPI_CLK starts toggling in the middle of the transfer (SPMODEx[CPx] = 0).



NOTE: Q = Undefined Signal.

**Figure 18-18. eSPI Transfer Format with SPMODEx[CPx] = 0**

Figure 18-19 shows the eSPI transfer format in which SPI_CLK starts toggling at the beginning of the transfer (SPMODEx[CPx] = 1).



NOTE: Q = Undefined Signal.

**Figure 18-19. eSPI Transfer Format with SPMODEx[CPx] = 1**

#### 18.3.1.8 CI and CP Values for Various eSPI Devices

1) Regular devices - eSPI mode0 - CI=CP=0

2) Regular devices - eSPI mode3 - CI=CP=1

For Winbond devices DO should also be set for dual output read command.

3) RapidS mode0 - CI=0, CP=1, HLD = 1

4) RapidS mode3 - CI=1, CP=0

## 18.4 eSPI Programming Examples

### 18.4.1 24-bit Address Example

The following sequence initializes the eSPI to read 36 bytes from 24-bit address memory, start address = 0x00_0040:

1. Configure a parallel I/O signal to operate as the eSPI CS1 output signal.
2. Write 0xFFFF_FFFF to SPIE to clear any previous events. Configure SPIM to enable all desired eSPI interrupts.
3. Configure SPMODE=0x8000_100F to enable normal operation, eSPI enabled.
4. Configure SPMODE1=0x2417_1108—REV1=1, PM1=4 (divide eSPI input clock by 10) , LEN1 = 7, POL1=1, CS1BEF=CS1AFT=CS1CG=1.
5. Configure SPCOM=0x0004_0027 so 4 bytes are skipped (1 for opcode and 3 for 24-bit address), TRANLEN= 36+4–1.
6. Configure SPITF=0x0300_0040—0x03 is read opcode while 0x000040 is the 24-bit start address.

### 18.4.2 16-bit Address Example

The following sequence initializes the eSPI to read 36 bytes from 16-bit address memory, start address = 0x0040:

1. Configure a parallel I/O signal to operate as the eSPI CS1 output signal.
2. Write 0xFFFF_FFFF to SPIE to clear any previous events. Configure SPIM to enable all desired eSPI interrupts.
3. Configure SPMODE=0x8000_100F to enable normal operation,eSPI enabled.
4. Configure SPMODE1=0x2417_1108—REV1=1, PM1=4 (divide eSPI input clock by 10) , LEN1 = 7, POL1=1, CS1BEF=CS1AFT=CS1CG=1.
5. Configure SPCOM=0x0003_0026 so 3 bytes are skipped (1 for opcode and 2 for 16-bit address), TRANLEN= 36+3–1.
6. Configure SPITF=0x0300_40xx (xx is don’t care)—0x03 is read opcode while 0x0040 is the 16-bit start address.

# Chapter 19
# SATA Controller

## 19.1 Overview

The serial ATA controller is a high-performance SATA solution incorporating some of the latest SATA-IO extensions. The SATA may also be referred to as a host bus adapter (HBA). The SATA controller is designed to operate in a system that supports command queuing and, in particular, a switching scheme based on a frame information structure (FIS) using port multipliers.

FIS-based switching requires the SATA controller to maintain in hardware a context for each command it has queued at the devices that are attached to it. FIS-based switching also requires the SATA controller to maintain a queue per attached device ensuring that the command issue order for each device is maintained. It can be used in SATA controllers, as well as storage area network (SAN), network attached storage (NAS), and RAID (redundant array of independent/inexpensive disks) devices.

SATA controller has the following features:

- Designed to comply with *Serial ATA 2.5 Specification*
- Supports speeds: 1.5 Gbps (first-generation SATA), 3 Gbps (second-generation SATA and eSATA)
- Supports advanced technology attachment packet interface (ATAPI) devices
- Contains high-speed descriptor-based DMA controller
- Supports native command queuing (NCQ) commands
- Supports port multiplier operation
- Supports hot plug including asynchronous signal recovery

Figure 19-1 shows a block diagram of SATA.

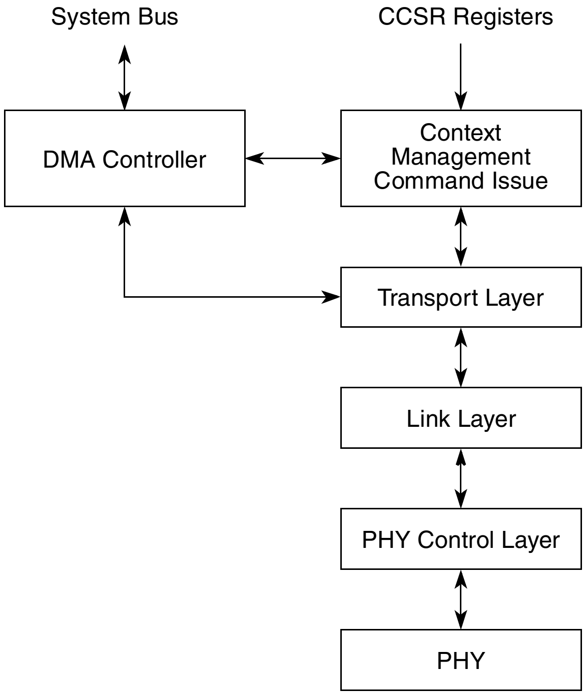


**Figure 19-1. SATA Block Diagram**

There are four layers in the SATA architecture: application, transport, link, and PHY. The application layer is responsible for overall ATA command execution, including controlling command block register accesses. The transport layer is responsible for placing control information and data to be transferred between the host and device in a packet/frame, known as a frame information structure (FIS). The link layer is responsible for taking data from the constructed frames, inserting control characters and moving data to the PHY layer. The PHY layer is responsible for 8B/10B encoding/decoding, then transmitting and receiving the encoded information as a serial data stream on the wire.

## 19.2 Command Operation

The SATA controller maintains a queue consisting of up to 16 commands. These commands can be distributed to a single attached device or, if the system contains a port multiplier, over each of the attached devices. It is possible to queue queued commands and non-queued commands into the SATA controller, provided the host software does not break protocol to any particular device (it is illegal to issue a non-queued command to a device that still has a queued command active as per ATAPI/ATA protocol).

### 19.2.1 Command Issue

When the host software is preparing to issue a command, it first builds a command descriptor as shown in Figure 19-28. The format of the command FIS is defined in the Serial ATA 2.5 standard shown in Figure 19-29. The software is also responsible for the creation of a scatter/gather list for data movement. This list should be defined to exactly match the transfer length as programmed into the command FIS. If the 16 entries are not sufficient, then an extended entry (ext) can be used to refer to an alternate table. When the descriptor is built, the host software locates a free command slot within the SATA controller by examining the command queue register. To issue the command, the host software programs the address of the command descriptor and the attributes into the appropriate command header locations and then issues the command by writing the PMP and setting the appropriate CQ bit in the command queue register.

### 19.2.2 Command Service

After a command is issued, the SATA controller takes ownership of the command descriptor, transferring the command FIS to the targeted device when required, servicing the data transfer using the scatter/gather list provided and transferring the status back into the command descriptor (if programmed).

### 19.2.3 Command Completion Interrupt Timing

When a command completes, it is possible to enable the SATA controller to generate an interrupt. Associated with some commands there will be a command completion status FIS. The SATA controller will always transfer the status FIS to memory whether it indicates an error or good command completion.

### 19.2.4 DMA Context (Read Data)

When receiving FIS's from attached devices, the SATA controller has to support interleaving from various devices. Data FIS's from device 0 could be interleaved with data FIS's for device 1. In order to accomplish this, the SATA controller maintains in hardware a context for each command which is pushed onto and pulled from the DMA controller when needed to service the transfers.

### 19.2.5 DMA Context (Write Data)

When the SATA controller receives an FIS indicating that the next operation to a particular device should be a data write transfer, the SATA controller will lock the interface by forcing the link layer to transition to X_RDY immediately and not go through idle SYNC. This will mean that write transfers will not have to be interleaved, which simplifies the transmit data path and eliminates the need for a complex scheduler.

### 19.2.6 DMAT Primitive Processing

The SATA controller supports the reception of the DMAT primitive. When the SATA controller receives a DMAT primitive from the device, it will perform the following actions.

The DMA controller will complete the current read burst and transfer the data to the transport layer FIFO. The DMA controller signals an EOF on the last data of the burst, which causes the link layer to insert the CRC and EOF. The context for this transfer is returned to the context store. Once this action is completed, the device can terminate the transfer or re-initiate the transfer as per Serial ATA Revision 2.5 Section 9.4.4.

## 19.3 Command Layer Overview

The function of the SATA command layer is to allow host software queue commands. It then manages the command issue and service using context to complete the queued commands.

### 19.3.1 SATA Memory Map/Register Definition

Table 19-1 shows the memory map for the SATA registers. The offsets to the memory map table are defined for both SATA hosts. That is, SATA1 starts at 0x1_8000 address offset and SATA2 at 0x1_9000. Undefined 4-byte address spaces within offset 0x000–0xFFF are reserved.

**NOTE**

All registers (except SYSPR) described in this section and descriptors described in Section 19.3.6, "Command Header," and Section 19.3.7, "Command Descriptor," use little-endian byte ordering. Software running on the local processor in big-endian mode must byte-swap the data.

In this table, and in the register figures and field descriptions, the following access definitions apply:

- Reserved fields are always ignored for the purposes of determining access type.
- 'R/W', 'R', and 'W' (read/write, read only, and write only) indicate that all the non-reserved fields in a register have the same access type.
- 'w1c' indicates that all of the non-reserved fields in a register are cleared by writing ones to them.
- 'Mixed' indicates a combination of access types.

**Table 19-1. SATA Register Summary**

| Offset | Register | Access | Reset Value | Section/Page |
|---|---|---|---|---|
| **SATA1—Block Base Address: 0x1_8000** | | | | |
| **SATA Command Registers** | | | | |
| 0x000 | CQR—Command queue register | R/W | 0x0000_0000 | 19.3.2.1/19-5 |
| 0x008 | CAR—Command active register | R | 0x0000_0000 | 19.3.2.2/19-6 |
| 0x010 | CCR—Command completed register | w1c | 0x0000_0000 | 19.3.2.3/19-6 |
| 0x018 | CER—Command error register | w1c | 0x0000_0000 | 19.3.2.4/19-7 |
| 0x020 | DE—Device error register | w1c | 0x0000_0000 | 19.3.2.5/19-8 |
| 0x024 | CHBA—Command header base address | R/W | 0x0000_0000 | 19.3.2.6/19-8 |
| 0x028 | HStatus—Host status register | w1c | 0x2000_0000 | 19.3.2.7/19-9 |
| 0x02C | HControl—Host control register | Mixed | 0x0000_0100 | 19.3.2.8/19-12 |
| 0x030 | CQPMP—Port number queue register | R/W | 0x0000_0000 | 19.3.2.9/19-13 |
| 0x034 | SIG—Signature register | R | 0xFFFF_FFFF | 19.3.2.10/19-14 |
| 0x038 | ICC—Interrupt coalescing control register | R/W | 0x0100_0000 | 19.3.2.11/19-14 |
| **SATA1 Superset Registers** | | | | |
| 0x100 | SStatus—SATA interface status register | R | 0x0000_0000 | 19.3.3.1/19-15 |
| 0x104 | SError—SATA interface error register | w1c | 0x0000_0000 | 19.3.3.2/19-16 |

Table 19-1. SATA Register Summary (continued)

| Offset | Register | Access | Reset Value | Section/Page |
|---|---|---|---|---|
| 0x108 | SControl—SATA interface control register | R/W | 0x0000_0300 | 19.3.3.3/19-18 |
| 0x10C | SNotification—SATA interface notification register | w1c | 0x0000_0000 | 19.3.3.4/19-19 |
| **SATA1 Control Status Registers** | | | | |
| 0x140 | TransCfg—Transport layer configuration | R/W | 0x0800_0016 | 19.3.4.1/19-20 |
| 0x144 | TransStatus—Transport layer status | R | 0x0000_0000 | 19.3.4.2/19-21 |
| 0x148 | LinkCfg—Link layer configuration | R/W | 0x0000_FF34 | 19.3.4.3/19-21 |
| 0x14C | LinkCfg1—Link layer configuration1 | R/W | 0x0000_0000 | 19.3.4.4/19-22 |
| 0x150 | LinkCfg2—Link layer configuration2 | R/W | 0x0000_0000 | 19.3.4.5/19-23 |
| 0x154 | LinkStatus—Link layer status | R | 0x0000_0000 | 19.3.4.6/19-23 |
| 0x158 | LinkStatus1—Link layer status1 | R | 0x0000_0000 | 19.3.4.7/19-24 |
| 0x15C | PhyCtrlCfg1—PHY control configuration1 | R/W | 0x0000_3800 | 19.3.4.8/19-26 |
| 0x160 | CommandStatus—Link layer command status | R | 0x0000_0000 | 19.3.4.9/19-27 |
| 0x164–<br>0x17C | Reserved | — | — | — |
| **SATA1 System Control Registers** | | | | |
| 0x410 | SYSPR—System priority register | R/W | 0x0000_0000 | 19.3.5.1/19-28 |
| 0x40C–<br>0xFFF | Reserved | — | — | — |
| SATA2—Block Base Address: 0x1_9000 | | | | |
| SATA2 has the same memory-mapped registers that are described for SATA1 from 0x1_8000 to 0x1_8FFF except the offsets are from 0x1_9000 to 0x1_9FFF. | | | | |

## 19.3.2 Command Registers

### 19.3.2.1 Command Queue Register (CQR)

Before queuing a command into the SATA controller, the CQR (shown in Figure 19-2) is first examined to detect a free command queue (CQ) slot. A free CQ slot is indicated by a 0 in a bit position. To queue a command, the bit corresponding to the CQ slot to use is set. At this point the SATA controller takes ownership of the command header space and command descriptor associated with the command slot. While the command is queued in the SATA controller or at the device, the command queue bit remains 1. When the command completes, this bit is cleared to 0 by the hardware. For a device error, the CQR holds the command queue bits at 1 for each command queued or issued to the device in error. When the host software clears the device error, the hardware in turn clears each of the commands queued.

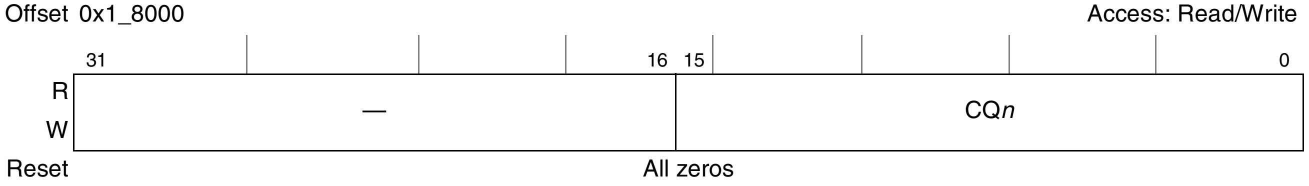


**Figure 19-2. Command Queue Register (CQR)**

Table 19-2 describes the CQR fields.

**Table 19-2. CQR Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 31–16 | — | Reserved |
| 15–0 | CQ*n* | Command *n* queue bit |

### 19.3.2.2 Command Active Register (CAR)

When a command is issued from the SATA controller to the device, the command is marked as active by the hardware setting the appropriate command active bit of the CAR (shown in Figure 19-3). Once a command completes, the hardware clears the appropriate bit of the CAR.

For a device error, the CAR holds the command active bits at 1 for each command issued to the device in error. When the host software clears the device error, the hardware in turn clears each of the commands queued.

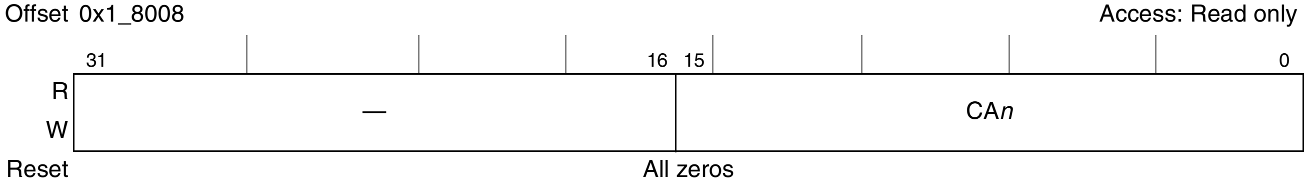


**Figure 19-3. Command Active Register (CAR)**

Table 19-3 describes the CAR fields.

**Table 19-3. CAR Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 31–16 | — | Reserved |
| 15–0 | CA*n* | Command *n* active bit |

### 19.3.2.3 Command Completed Register (CCR)

When a command completes, the hardware sets the command completed bit for that command in the CCR (shown in Figure 19-4) to a 1. The hardware also clears both the command queue and the command active bit for that command. When the software needs to acknowledge the reception of the command complete, it can do so in two ways:

- Writing a 1 to the command complete bit

- Issuing a command to the command slot

An interrupt coalescing scheme runs on the CCR. When the register contains a value other than 0x0000_0000, an interrupt coalescing timer runs. Each time a command completion is acknowledged, the timer is reset. When the timer times out, an interrupt is generated.

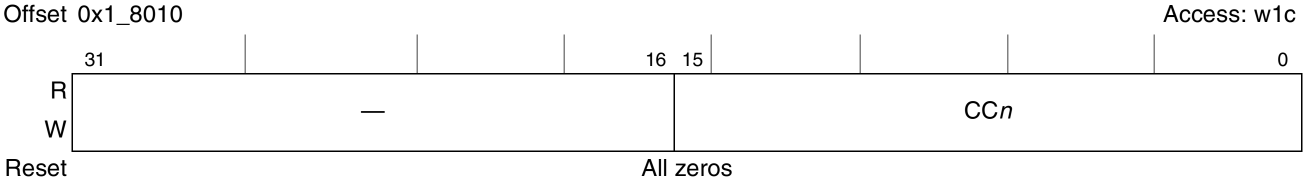


**Figure 19-4. Command Completed Register (CCR)**

Table 19-4 describes the CCR fields.

**Table 19-4. CCR Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 31–16 | — | Reserved |
| 15–0 | CC*n* | Command *n* completed bit |

### 19.3.2.4 Command Error Register (CER)

When a device errors a command by setting the error bit in the status register, this is detected by the SATA controller as a single device error. The associated command completing due to error is indicated by the hardware setting the command error bit for that command in the CER (shown in Figure 19-5). For safe operation under both command queuing and non-queuing operation, all commands queued into the SATA controller and at the device are considered aborted. The queue for that device is stopped. The values of the registers CQR, CAR, and CCR will allow the host software to know which commands have completed without error and those that were queued at the SATA controller and at the device.

When the host software clears the device error (by writing 1 to DER), the software is also responsible to clear CER by writing a 1 to the command error bit for the command that was in error. After the error condition at the device has been cleared, the host application software can reissue the commands to the SATA controller, which were aborted on the reception of the single device error.

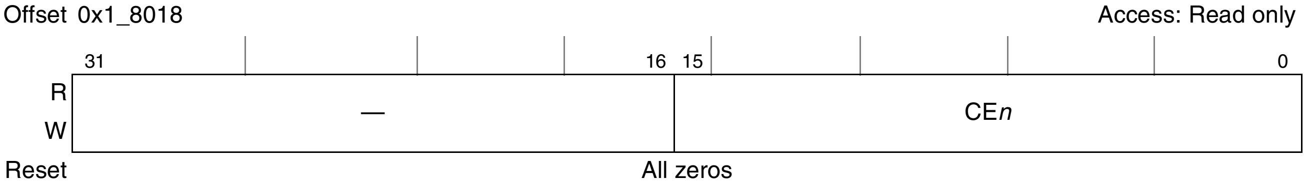


**Figure 19-5. Command Error Register (CER)**

Figure 19-5 describes the CER fields.

**Table 19-5. CER Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 31–16 | — | Reserved |
| 31–0 | CE*n* | Command *n* error bit |

### 19.3.2.5 Device Error Register (DER)

When a single device error is detected, the device that issued the error is indicated by the hardware setting the device error bit to a 1 in the DER (shown in Figure 19-6). The procedure as outlined in the command error register applies to the queues and to restarting the device.

The host application software acknowledges the device in error by clearing the device error bit. The device error is cleared by writing a 1 to the appropriate device error bit. When this action is performed, the queue to the device in error is cleared and is ready to have commands queue.

While a device is in error, no command can be queued for that device.

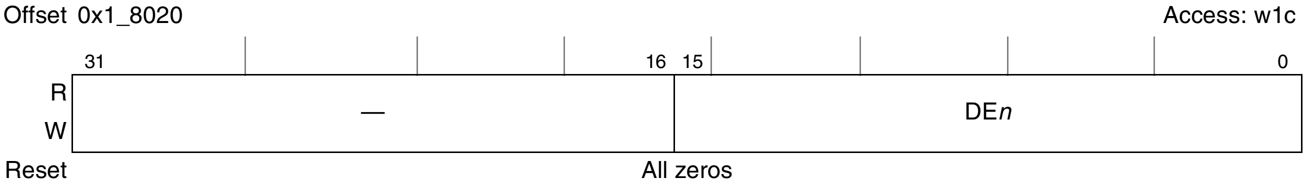


**Figure 19-6. Device Error Register (DER)**

Table 19-6 describes the DER fields.

**Table 19-6. DER Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 31–16 | — | Reserved |
| 15–0 | DE*n* | Device *n* error bit |

### 19.3.2.6 Command Header Base Address Register (CHBA)

The CHBA is shown in Figure 19-7. This holds the address in memory of where the command header block is located. It must be written as part of the host software initialization process. After the SATA controller hardware is brought online, the SATA controller takes ownership of this register. The address in this register should not be changed while the SATA controller is online.

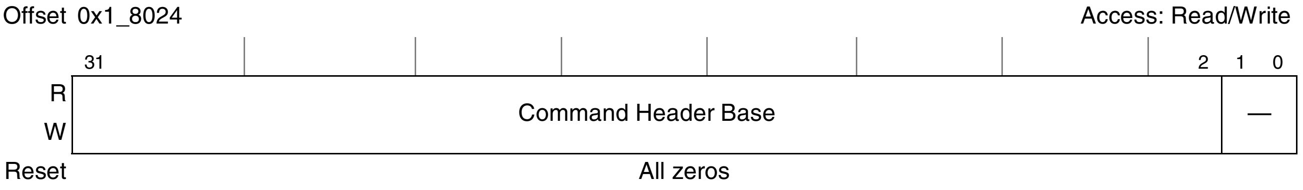


**Figure 19-7. Command Header Base Address Register (CHBA)**

Table 19-7 describes the CHBA fields.

**Table 19-7. CHBA Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 31–2 | CHBA | Command header base address |
| 1–0 | — | Reserved, should be cleared. |

### 19.3.2.7 Host Status Register (HStatus)

HStatus, shown in Figure 19-8, holds the status of the SATA controller as well as the interrupt sources. When an event occurs, the interrupt bit is set regardless of the status of the associated interrupt enable bit. The interrupt signal from the SATA controller is gated with the associated interrupt enable register. For all interrupt bits other than the interrupt on command complete bit, when software has processed the interrupt condition, it acknowledges the interrupt by writing a 1 to the interrupt source bit. This action will clear the interrupt signal if there are no other outstanding interrupts in HStatus.

The interrupt on command complete requires special processing. This bit is set as a result of the programmed interrupt coalescing algorithm running on the register CCR contents. For the interrupt on command complete bit, the command(s) that have completed to cause this interrupt need to be cleared by clearing the command N completed bit of the CCR. When the number or staleness of the CCR falls below the programmed interrupt coalescing algorithm, the interrupt on command complete bit clears.

Offset 0x1_8028 Access: w1c

| | 31 | 30 | 29 | 28 | | | | | | | | | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | HS_ON | HS_OFF | BE | — | | | | | | | | | | ME | — | |
| Reset | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | | DUE | DOE | CET | CER | FOT | FOR | — | | FE | PR | SIGU | SNTFU | DE | CC |
| Reset | All zeros | | | | | | | | | | | | | | | |

**Figure 19-8. Host Status Register (HStatus)**

Table 19-8 describes the HStatus fields.

**Table 19-8. HStatus Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 31 | HS_ON | Online/offline. This bit indicates if the SATA controller is online or offline.<br>0 Offline. The SATA controller is non-operational and the PHY is held in reset.<br>1 Online. The SATA controller is operational. |
| 30 | HS_OFF | Going offline. This bit indicates that the SATA controller is going offline it is waiting for the commands queued within the SATA controller or active at the device to complete.<br>0 Host is not in process going offline<br>1 Host is in process going offline |

Table 19-8. HStatus Field Descriptions (continued)

| Bit | Name | Description |
|---|---|---|
| 29 | BE | BIST error. When the protocol is placed into BIST this bit maps the BIST error.<br>0 Indicates the link layer is passing BIST<br>1 Indicates that the link layer is not passing BIST<br>When the protocol is not in BIST this bit will assert high and can be ignored. |
| 28–24 | — | Reserved |
| 23–19 | — | Reserved |
| 18 | ME | SATA controller master error. Indicates if the host received an error on the system bus interface during the access to memory.<br>0 No error response is received when a transfer was made into the memory<br>1 Error response is received during the transfer into the memory |
| 17–16 | — | Reserved |
| 15–14 | — | Reserved |
| 13 | DUE | Data underrun.<br>0 No underrun encountered (data was retrieved from external memory in time to send a complete FIS)<br>1 The SATA controller encountered an underrun condition while sending the FIS |
| 12 | DOE | Data overrun.<br>0 No overrun condition encountered<br>1 The SATA controller encountered an overrun condition while receiving the FIS |
| 11 | CET | CRC error Tx. When set, this bit indicates that one or more CRC errors occurred in Tx data path. |
| 10 | CER | CRC error Rx. When set, this bit indicates that one or more CRC errors occurred in Rx data path. |
| 9 | FOT | FIFO overflow Tx. When set, this bit indicates that Tx FIFO is in overflow condition while sending FIS. |
| 8 | FOR | FIFO overflow Rx. When set, this bit indicates that Rx FIFO is in overflow condition while receiving FIS. |
| 7–6 | — | Reserved |
| 5 | FE | Fatal error. When set, this bit indicates that fatal error occurred in SATA controller. In this state, the interrupt will be generated if FATAL_INT is set in the host control register. Write ‘1’ to clear the interrupt source. |
| 4 | PR | PHY ready. When set, this bit indicates that PHY READY signal was changed. In this state, the interrupt will be generated if PHYRDY_INT is set in the host control register. Write ‘1’ to clear the interrupt source. |
| 3 | SIGU | Signature update. When set, this bit indicates that the signature is updated in the host signature register. In this state, the interrupt will be generated if SIG_INT is set in the host control register. Write ‘1’ to clear the interrupt source. |
| 2 | SNTFU | SNotification update. When set, this bit indicates that the SNotification register has at least one bit set. In this state, the interrupt will be generated if SNTFY_INT is set in the host control register. Write ‘1’ to clear the interrupt source. |

Table 19-8. HStatus Field Descriptions (continued)

| Bit | Name | Description |
|---|---|---|
| 1 | DE | Device error. When set, this bit indicates that the DE register has at least one bit set. In this state, the interrupt will be generated if DE_INT is set in the host control register. Write ‘1’ to clear the interrupt source. |
| 0 | CC | Command complete. When set, this bit indicates that the register CCR has at least one bit set. In this state, the interrupt will be generated if CC_INT is set in the host control register. Write ‘1’ to clear the interrupt source. |

#### 19.3.2.7.1 Error Processing

On single device error:

1. Examine the register DER to determine which device is in error state. There might be multiple devices in error.
2. Examine the register CER to determine which command was in error. The software knows which command belongs to which device.
3. Examine the status location of the descriptor of the command in error and determine the reason for the error.
4. If needed, the software should send commands to the device to clear down the error condition on device or for further examination of the device's status.
5. Clear the DER*n* bit by writing 1 to bit *n*, where *n* indicates the device in error. This will also clear out the outstanding commands for that device.
6. Clear the CER*n* bit by writing 1 to bit *n*, where *n* indicates the associated command in error. After that, the software can reissue command to the device if needed.

On fatal error:

1. Read the error register and other registers to determine how many commands are outstanding and how many have completed without error.
2. Bring the SATA controller offline. When this happens all queues within the SATA controller will be cleared.
3. Perform what corrective action the software determines is necessary.
4. Bring the SATA controller online. This will cause an out-of-bounds (OOB) to be run at the PHY level which will clear down any attached device.

### 19.3.2.8 Host Control Register (HControl)

HControl, shown in Figure 19-9, is written to control the operation of the SATA controller. To enable an interrupt, the associated bit must be set; to disable the interrupt, the associated bit must be cleared.

Offset 0x1_802C Access: Mixed

| | 31 | 30 | 29–16 |
|---|---|---|---|
| R | HC_ON | HC_OFF | — |
| W | | | |
| Reset | All zeros | | |

| | 15–11 | 10 | 9 | 8–6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| R | — | SNOOP_EN | PM_EN | — | FATAL_INT | PHYRDY_INT | SIG_INT | SNTFY_INT | DE_INT | CC_INT |
| W | | | | | | | | | | |
| Reset | 0 0 0 0 0 | 0 | 0 | 1 0 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Figure 19-9. Host Control (HControl) Register**

Table 19-9 describes the HControl fields.

**Table 19-9. HControl Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 31 | HC_ON | Online/offline.<br>0 Offline. Bring the SATA controller offline and place the PHY in reset<br>1 Online. Bring the SATA controller online |
| 30 | HC_OFF | Offline request status.<br>1 The SATA controller is currently completing an operation and will go offline when the operation completes. When this bit is set, the SATA controller can be forced to go offline by aborting its current operation by writing 0 to the HC_ON bit. |
| 29–15 | — | Reserved |
| 14–12 | — | Reserved. Reset value must be preserved when writing to the register. |
| 11 | — | Reserved |
| 10 | SNOOP_EN | Snoop enable during header fetch.<br>0 Snoop not enabled during command header fetch.<br>1 Snoop enabled during command header fetch. |
| 9 | PM_EN | Port multiplier attached. This bit is used to indicate if the HBA is attached to a port multiplier. This bit is set or cleared by software.<br>0 This SATA controller is directly attached to a SATA device. The SATA controller hardware does not auto-detect the presence of a port multiplier; this is to allow for future changes in signature type for the port multiplier.<br>1 A port multiplier is attached to the SATA controller. |
| 8–6 | — | Reserved. Reset value must be preserved when writing to the register. |
| 5 | FATAL_INT | Enable interrupt on fatal error. |
| 4 | PHYRDY_INT | Enable interrupt on PHY ready change. |
| 3 | SIG_INT | Enable interrupt signature update. |
| 2 | SNTFY_INT | Enable interrupt on SNotify register update. |

Table 19-9. HControl Field Descriptions (continued)

| Bit | Name | Description |
|---|---|---|
| 1 | DE_INT | Enable interrupt on single device error. |
| 0 | CC_INT | Enable interrupt on command complete. |

#### 19.3.2.8.1 Bringing the SATA Controller Online/Offline

This HC_ON bit in HControl allows the host software to bring the SATA controller online or offline. The SATA controller online status should only be changed when there are no commands queued in the SATA controller or at any attached device.

When the host application wishes to bring the SATA controller offline it clears the HC_ON control bit. This acts as a request to the SATA controller to go offline. The SATA controller will signal it has completed this operation by clearing the HS_ON bit in the HStatus register. If any commands are outstanding at SATA controller or device then the SATA controller will wait for the operation to complete before going offline.

It the host application wishes to bring the SATA controller offline regardless of the queue status, it clears the HC_ON bit while the HS_OFF bit of the HStatus register is set.

When the host application wishes to bring the SATA controller online, it sets the HC_ON control bit. This acts as a request to the SATA controller to go online. The SATA controller will signal it has completed this operation by setting to 1 the HS_ON status bit.

### 19.3.2.9 Port Number Queue Register (CQPMP)

When queuing a command into the SATA controller, the CQPMP, shown in Figure 19-10, is written with the value of the PMP field that addresses the device to which the command will be issued. If the device is directly attached (that is, there is no port multiplier in the system), then this register is not required and should be cleared.

Offset 0x1_8030 Access: Read/Write

| | 31 … 4 | 3 … 0 |
|---|---|---|
| R<br>W | — | CQPMP |
| Reset | All zeros | |

Figure 19-10. Port Number Queue Register (CQPMP)

Table 19-10 describes the CQPMP fields.

Table 19-10. CQPMP Field Descriptions

| Bit | Name | Description |
|---|---|---|
| 31–4 | — | Reserved |
| 3–0 | CQPMP | Command queue port multiplier field |

### 19.3.2.10 Signature Register (SIG)

The 32-bit SIG register, shown in Figure 19-11, contains the initial signature of an attached device when the first D2H register FIS is received from that device.

Offset 0x1_8034 Access: Read only

| | 31–24 | 23–16 | 15–8 | 7–0 |
|---|---|---|---|---|
| R<br>W | LBA_HIGH | LBA_MID | LBA_LOW | SEC_CNT |
| Reset | All ones | | | |

**Figure 19-11. Signature Register (SIG)**

Table 19-11 describes the SIG register fields.

**Table 19-11. SIG Register Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 31–24 | LBA_HIGH | LBA high register |
| 23–16 | LBA_MID | LBA mid register |
| 15–8 | LBA_LOW | LBA low register |
| 7–0 | SEC_CNT | Sector count register |

### 19.3.2.11 Interrupt Coalescing Control Register (ICC)

When a command completes, the SATA controller sets the corresponding bit in the command completed register. The interrupt coalescing scheme runs on the SIG register, shown in Figure 19-12. The scheme runs in two ways:

- If the number of completed commands exceeds the threshold, then the interrupt will be signaled.
- If any command complete bit has been set in the register for a number of HCLK ticks equal to the programmed count, then the interrupt will be set. This timer will be reset each time a command completion is acknowledged by the application layer software.

Offset 0x1_8038 Access: Read/Write

| | 31–29 | 28–24 | 23–19 | 18–0 |
|---|---|---|---|---|
| R<br>W | — | ITC | — | ITTCV |
| Reset | 0 0 0 | 0 0 0 0 1 | 0 0 0 0 0 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 |

**Figure 19-12. Interrupt Coalescing Control Register (ICC)**

Table 19-12 describes the ICC fields.

**Table 19-12. ICC Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 31–29 | — | Reserved |
| 28–24 | ITC | Interrupt threshold count. The number of command completions that will raise the interrupt.<br>00000Implied no threshold and the interrupt will be signaled based on the threshold timer.<br>00001, 01111The number of command complete bits which, if set, will cause the interrupt to be signaled. |
| 23–19 | — | Reserved |
| 18–0 | ITTCV | Interrupt threshold timer compare value. The number of AHB ticks for which a command complete bit has to be set before the interrupt will be signaled. A value of 0 indicates that whenever a command complete bit is set the interrupt should be signaled. |

## 19.3.3 SATA Superset Registers

Serial ATA provides an additional block of registers to control the interface and to retrieve interface state information.

### 19.3.3.1 SATA Interface Status Register (SStatus)

SStatus, shown in Figure 19-13, is a 32-bit read-only register that conveys the current state of the interface and host adapter. The register conveys the interface state at the time it is read and is updated continuously and asynchronously by the host adapter. Writes to this register have no effect.

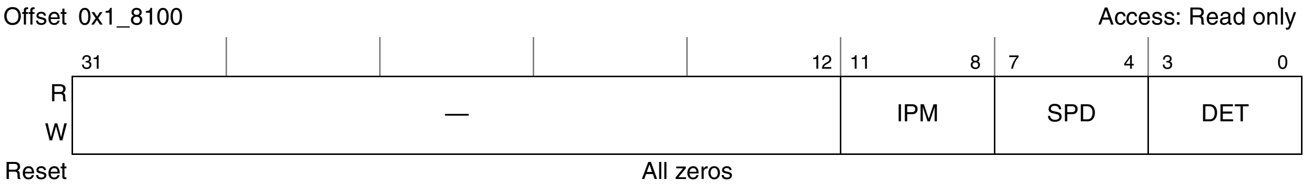


**Figure 19-13. SATA Interface Status Register (SStatus)**

Table 19-13 describes the SStatus fields.

**Table 19-13. SStatus Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 31–12 | — | Reserved |
| 11–8 | IPM | Interface power management state. Indicates the current interface power management state.<br>0000 Device not present or communication not established<br>0001 Interface in active state<br>0010 Interface in partial power management state<br>0110 Interface in slumber power management state<br>All other values reserved |

Table 19-13. SStatus Field Descriptions (continued)

| Bit | Name | Description |
|---|---|---|
| 7–4 | SPD | Speed. Indicates the negotiated interface communication speed established.<br>0000 No negotiated speed (device not present or communication not established)<br>0001 First-generation communication rate negotiated<br>0010 Second-generation communication rate negotiated<br>All other values reserved |
| 3–0 | DET | Detection. Indicates the interface device detection and PHY state.<br>0000 No device detected and PHY communication not established<br>0001 Device presence detected but PHY communication not established<br>0011 Device presence detected and PHY communication established<br>0100 PHY in offline mode as a result of the interface being disabled or running in a BIST loopback mode<br>All other values reserved |

### 19.3.3.2 SATA Interface Error Register (SError)

SError, shown in Figure 19-14, is a 32-bit register that conveys supplemental interface error information to complement the error information available in the shadow register block error register. The register represents all the detected errors accumulated since the last time the SError register was cleared (whether recovered by the interface or not). Set bits in the error register are explicitly cleared by a write operation to the SError register or by a reset operation. The error bits that have been set in this register are cleared by writing a 1 to the corresponding field. Host software should clear the interface SError register at appropriate checkpoints in order to best isolate error conditions and the commands they impact.

Bits 31–16 of this register represent the DIAG decode bits, which contain diagnostic error information, for use by diagnostic software in validating correct operation or isolating failure modes. Bits 15–0 represent the ERR decode bits, which contain information for use by the host software in determining the appropriate response to the error condition.

Offset 0x1_8104 Access: w1c

DIAG Decode (bits 31–16) ERR Decode (bits 15–0)

| | 31–28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15–12 | 11 | 10 | 9 | 8 | 7–2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | — | A | X | F | — | S | H | C | D | B | W | IN | N | — | E | — | C | T | — | M | ITG |
| W | | w1c | w1c | w1c | | w1c | w1c | w1c | w1c | w1c | w1c | w1c | w1c | | w1c | | w1c | w1c | | w1c | w1c |
| Reset | All zeros | | | | | | | | | | | | | | | | | | | | |

Figure 19-14. SATA Interface Error Register (SError)

Table 19-14 describes the SError field descriptions.

Table 19-14. SError Field Descriptions

| Bit | Name | Description |
|---|---|---|
| **DIAG Decode** | | |
| 31–28 | — | Reserved bit for future use. Should be cleared. |

Table 19-14. SError Field Descriptions (continued)

| Bit | Name | Description |
|---|---|---|
| 27 | A | Port selector presence detected. This bit is set when COMWAKE is received while the host is in state HP2: HR_AwaitCOMINIT. On power-up reset this bit is cleared. The bit is cleared when the host writes a 1 to this bit location. |
| 26 | X | Exchanged. When set to 1 this bit indicates that device presence has changed since the last time this bit was cleared. The means by which the implementation determines that the device presence has changed is vendor specific. This bit may be set anytime a PHY reset initialization sequence occurs as determined by reception of the COMINIT signal, whether in response to a new device being inserted, to a COMRESET having been issued, or to power-up. |
| 25 | F | Unrecognized FIS type. When set to 1, this bit indicates that since the bit was last cleared, one or more FIS's were received by the transport layer with good CRC, but they had a type field that was not recognized. |
| 24 | — | Reserved. |
| 23 | S | Link sequence error. When set to 1, this bit indicates that one or more link state machine error conditions were encountered since the last time this bit was cleared. The link layer state machine defines the conditions under which the link layer detects an erroneous transition. |
| 22 | H | Handshake error. When set to 1, this bit indicates that one or more R_ERRP handshake responses were received in response to frame transmission. Such errors may be the result of a CRC error detected by the recipient, of a disparity or 10b/8b decoding error, or of other error conditions leading to a negative handshake on a transmitted frame. |
| 21 | C | CRC error. When set to 1, this bit indicates that one or more CRC errors occurred with the link layer since the bit was last cleared. |
| 20 | D | Disparity error. When set to 1, this bit indicates that incorrect disparity was detected one or more times since the last time the bit was cleared. |
| 19 | B | 10b to 8b decode error. When set to 1, this bit indicates that one or more 10-bit to 8-bit decoding errors occurred since the bit was last cleared. |
| 18 | W | COMWAKE detected. When set to 1, this bit indicates that a COMWAKE signal was detected by the PHY since the last time this bit was cleared. |
| 17 | IN | PHY internal error. When set to 1, this bit indicates that the PHY detected some internal error since the last time this bit was cleared. |
| 16 | N | PHYRDY change. When set to 1, this bit indicates that the PHYRDY signal changed state since the last time this bit was cleared. |
| **ERR Decode** | | |
| 15–12 | — | Reserved bit for future use; should be cleared. |
| 11 | E | E Internal error. The host bus adapter experienced an internal error that caused the operation to fail and may have put the host bus adapter into an error state. Host software should reset the interface before retrying the operation. If the condition persists, the host bus adapter may suffer from a design issue rendering it incompatible with the attached device. |
| 10 | — | Reserved. |

Table 19-14. SError Field Descriptions (continued)

| Bit | Name | Description |
|---|---|---|
| 9 | C | Non-recovered persistent communication or data integrity error. A communication error that was not recovered occurred that is expected to be persistent. Because the error condition is expected to be persistent, the operation need not be retried by the host software. Persistent communications errors may arise from faulty interconnect with the device, from a device that has been removed or has failed, or a number of other causes. |
| 8 | T | Non-recovered transient data integrity error: A data integrity error occurred that was not recovered by the interface. Because the error condition is not expected to be persistent, the operation should be retried by the host software. |
| 7–2 | — | Reserved |
| 1 | M | Recovered communications error. Communications between the device and host were temporarily lost but were re-established. This can arise from a device temporarily being removed, from a temporary loss of PHY synchronization, or from other causes, and may be derived from the PHYRDY*n* signal between the PHY and link layers. No action is required by the host software, because the operation ultimately succeeded. However, the host software may elect to track such recovered errors to gauge overall communications integrity and potentially step down the negotiated communication speed. |
| 0 | ITG | Recovered data integrity error. A data integrity error occurred that was recovered by the interface through a retry operation or other recovery action. This can arise from a noise burst in the transmission, a voltage supply variation, or other causes. No action is required by host software, because the operation ultimately succeeded. However, the host software may elect to track such recovered errors to gauge overall communications integrity and potentially step down the negotiated communication speed. |

### 19.3.3.3 SATA Interface Control Register (SControl)

SControl, shown in Figure 19-15, is a 32-bit read-write register that provides the interface by which software controls SATA interface capabilities. Writes to the SControl register result in an action being taken by the host adapter or interface. Reads from the register return the last value written to it.

Offset 0x1_8108 Access: Read/Write

| | 31 | 21 20 19 18 17 16 | 15 12 | 11 8 | 7 4 | 3 0 |
|---|---|---|---|---|---|---|
| R<br>W | — | | SPM | IPM | SPD | DET |
| Reset | 0 0 0 0 0 0 0 0 0 0 0 | 0 0 0 0 0 | 0 0 0 0 | 0 0 1 1 | 0 0 0 0 | 0 0 0 0 |

Figure 19-15. SATA Interface Control Register (SControl)

Table 19-15 describes the SControl fields.

**Table 19-15. SControl Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 31–16 | — | Reserved, should be cleared. |
| 15–12 | SPM | Select power management. Used to select a power management state. A non-zero value written to this field will cause the power management state specified to be initiated. A value written to this field is treated as a one-shot.<br>0000 No power management state transition requested<br>0001 Transition to the partial power management state initiated<br>0010 Transition to the slumber power management state initiated<br>0100 Transition to the active power management state initiated<br>All other values reserved |
| 11–8 | IPM | Interface power management. The enabled interface power management states can be invoked via the SATA interface power management capabilities.<br>0000 No interface power management state restrictions<br>0001 Transitions to the partial power management state disabled<br>0010 Transitions to the slumber power management state disabled<br>0011 Transitions to both the partial and slumber power management states disabled<br>All other values reserved |
| 7–4 | SPD | Speed. Highest allowed communication speed the interface is allowed to negotiate when interface communication speed is established.<br>0000 No speed negotiation restrictions<br>0001 Limit speed negotiation to a rate not greater than first-generation communication rate<br>0010 Limit speed negotiation to a rate not greater than second-generation communication rate<br>All other values reserved |
| 3–0 | DET | Detection. Controls the host adapter device detection and interface initialization.<br>0000 No device detection or initialization action requested<br>0001 Perform interface communication initialization sequence to establish communication. This is functionally equivalent to a hard reset and results in the interface being reset and communications re-initialized. Upon a write to the SControl register that sets the DET field to 0001, the host interface should transition to the HP1: HR_Reset [Delete space after state and should remain in that state until the DET field is set to a value other than 0001 by a subsequent write to the SControl register.<br>0100 Disable the SATA interface and put PHY in offline mode<br>All other values reserved |

### 19.3.3.4 SATA Interface Notification Register (SNotification)

SNotification, shown in Figure 19-16, is a 32-bit, write-one-to-clear register that conveys the devices that have sent the host a set device bits FIS with the notification bit. When the host receives a set device bits FIS with the notification bit set to 1, the host should set the bit in SNotification corresponding to the value of the PM port field in the received FIS. For example, if the PM port field is set to 7 then the host should clear bit 7 by writing a 1 to it. Next, the host should generate an interrupt if the I bit of the set device bits FIS is set to 1 and interrupts are enabled.

In this register, bits previously set are explicitly cleared by a write operation or by a power-on-reset operation. If the register is not cleared due to a COMRESET, the software is responsible for clearing the register as appropriate.

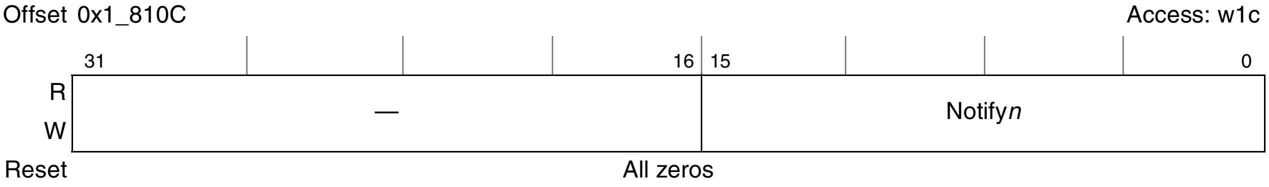


**Figure 19-16. SATA Interface Notification Register (SNotification)**

Table 19-16 describes the SNotification fields.

**Table 19-16. SNotification Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 31–16 | — | Reserved, should be cleared. |
| 15–0 | Notify*n* | Represents whether a particular device with the corresponding PM port number *n* has sent a set device bits FIS to the host with the notification bit set. |

## 19.3.4 Control Status Registers

### 19.3.4.1 Transport Layer Configuration Register (TransCfg)

TransCfg, shown in Figure 19-17, controls the configuration of the transport layer.

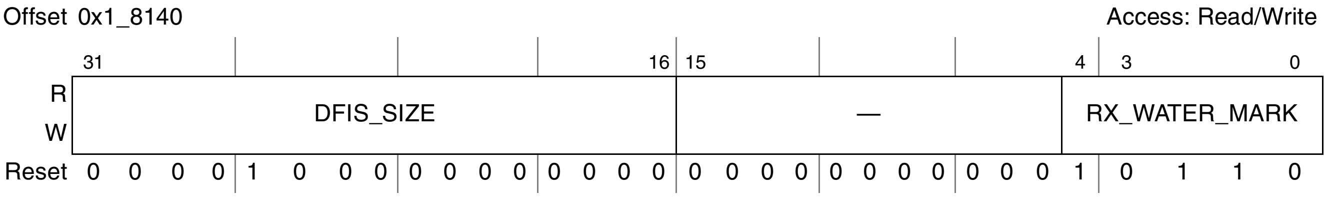


**Figure 19-17. Transport Layer Configuration Register (TransCfg)**

Table 19-17 describes the TransCfg fields.

**Table 19-17. TransCfg Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 31–16 | DFIS_SIZE | Data FIS framing length words.   Determines the maximum length each data FIS should be. |
| 15–5 | — | Reserved |
| 4–0 | RX_WATER_MARK | This sets the number of locations in the 58-deep Rx FIFO that can be used before the transport layer instructs the link layer to transmit HOLDS to the transmitting end. Note that it can take some time for the HOLDs to get to the other end, and that in the interim there must be enough room in the FIFO to absorb all data that could arrive. An initial value of 22 is recommended. |

### 19.3.4.2 Transport Layer Status Register (TransStatus)

TransStatus, shown in Figure 19-18, can be read to determine the status of the transport layer.

Offset 0x1_8144 Access: Read only

| | 15–8 | 7–0 |
|---|---|---|
| R<br>W | TX_SM | RX_SM |
| Reset | | |

**Figure 19-18. Transport Layer Status Register (TransStatus)**

Table 19-18 describes the TransStatus fields.

**Table 19-18. TransStatus Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 15–8 | TX_SM | Indicates the state of Tx state machine. |
| 7–0 | RX_SM | Indicates the state of Rx state machine. |

### 19.3.4.3 Link Layer Configuration Register (LinkCfg)

LinkCfg, shown in Figure 19-19, controls the configuration of the link layer.

Offset 0x1_8148 Access: Read/Write

| | 31–27 | 26 | 25–16 |
|---|---|---|---|
| R<br>W | — | POE | PRT |
| Reset | All zeros | | |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | AR | | | | | | | | EPNRT | S4A | RX_SCR_EN | TX_SCR_EN | TX_PRIM_JUNK | TX_CONT_EN | RX_BAD_CRC | TX_BAD_CRC |
| Reset | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |

**Figure 19-19. Link Layer Configuration Register (LinkCfg)**

Table 19-19 describes the LinkCfg fields.

**Table 19-19. LinkCfg Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 31–27 | — | Reserved |
| 26 | POE | Primitive override enable. When set, this bit enables the replacement of a single primitive, as specified by CFG_PRIM/CFG_CD, when the link layer state machine is in the CFG_PRIM_OVR_STATE state. This bit has to be toggled from a 0 to a 1 to enable this feature. |

**Table 19-19. LinkCfg Field Descriptions (continued)**

| Bit | Name | Description |
|---|---|---|
| 25–16 | PRT | PHY ready timer. These ten bits specify the timeout value of the PHY_READY timer. If EN_PHY_TO is set, the link layer will count down on every rising edge of scanTxClk, as long as PHY_READY is de-asserted. When the counter reaches 0, a PHY_RESET will be issued to the PHY to try and re-establish communications with the far end. The timer is initially loaded with a value equal to the concatenation of {PHY_READY_TIMER, 9b0_0000_0000}. |
| 15–8 | AR | Align insertion rate. The SATA specification requires that the link layer send a pair of ALIGN primitives at least every 254 words of data. This is achieved by setting ALIGN_RATE to ‘11111111’. However, for test purposes it is possible to send ALIGNs at a higher rate. This can be achieved by setting ALIGN_RATE to a lower value (that is, ALIGN_RATE-1); words will be sent by the link layer between each set of ALIGN primitive pairs.<br>**Note:** If SEND_4_ALIGNS is set, one should not set the ALIGN_RATE to be four or less. If SEND_4_ALIGNS is not set, one should not set the ALIGN_RATE to be two or less. |
| 7 | EPNRT | Enable PHY not ready timer. If PHY_READY is de-asserted for a length of time, as specified by CFG_PHY_READY_TIMER, then this bit, when asserted, enables the link layer to re-issue a PHY_RESET, thereby re-initiating OOB. |
| 6 | S4A | Send four ALIGNs. When asserted, four ALIGN primitives are transmitted at the specified rate, instead of the normal two ALIGNS. |
| 5 | RX_SCR_EN | Rx scramble enable. If this bit is asserted, then descrambling of the receive data is enabled as per the SATA specification. |
| 4 | TX_SCR_EN | Tx scramble enable. If this bit is asserted, then scrambling of the transmit data is enabled as per the SATA specification. |
| 3 | TX_PRIM_JUNK | TX prim junk. If this bit is de-asserted, then scrambled junk data is sent after a CONT primitive, as per the SATA specification. If this bit is asserted, then the single character 0xDEADBEEF is sent continuously instead. This is to aid debug. |
| 2 | TX_CONT_EN | TX CONT. If this bit is asserted, then the transmission of CONT primitives is enabled. If de-asserted, then long sequences of repeated primitives can be sent by the link layer. |
| 1 | RX_BAD_CRC | RX bad CRC. When a rising edge is detected on this bit, it causes a bad CRC to be detected for the current frame. This bit has to be toggled from a 0 to a 1 to enable this feature. |
| 0 | TX_BAD_CRC | Tx bad CRC. A bad CRC (inverted value of the correct CRC) value will be transmitted for one FIS only by the link layer when a rising edge is detected on this signal. This bit has to be toggled from a 0 to a 1 to enable this feature. |

### 19.3.4.4 Link Layer Configuration Register1 (LinkCfg1)

LinkCfg1, shown in Figure 19-20, controls the configuration of the link layer.

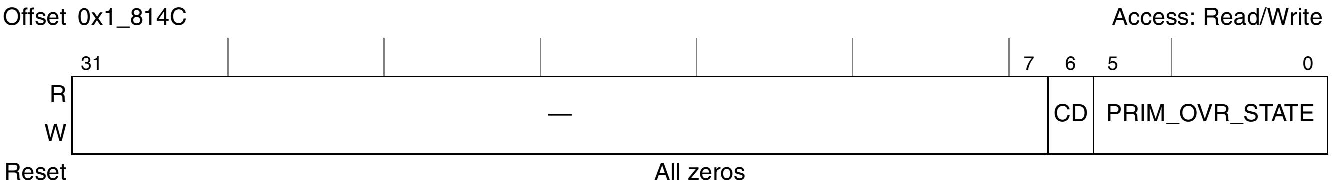


**Figure 19-20. Link Layer Configuration Register1 (LinkCfg1)**

Table 19-20 describes the LinkCfg1 fields.

Table 19-20. LinkCfg1 Field Descriptions

| Bit | Name | Description |
|---|---|---|
| 31–7 | — | Reserved |
| 6 | CD | This bit specifies whether the data used during the primitive override should be a data character or a primitive. For example, if CD = 1, PRIM_OVR_STATE = L_SendEOF and PRIM = WTRM, then a WTRM primitive will be inserted into the datastream instead of an EOF (whenever a rising edge is seen on PRIM_OVR_EN). If CD = 0, then a normal data character (as specified by PRIM) is inserted into the datastream instead of the EOF. |
| 5–0 | PRIM_OVR_STATE | Prim override state. These 6 bits are used in the primitive override debug functionality. When the link layer detects a positive edge on PRIM_OVR_EN, it overrides the next primitive that would be inserted during the PRIM_OVR_STATE, with the data specified by the PRIM and CD configuration bits. |

## 19.3.4.5 Link Layer Configuration Register2 (LinkCfg2)

LinkCfg2, shown in Figure 19-21, controls the configuration of the link layer.

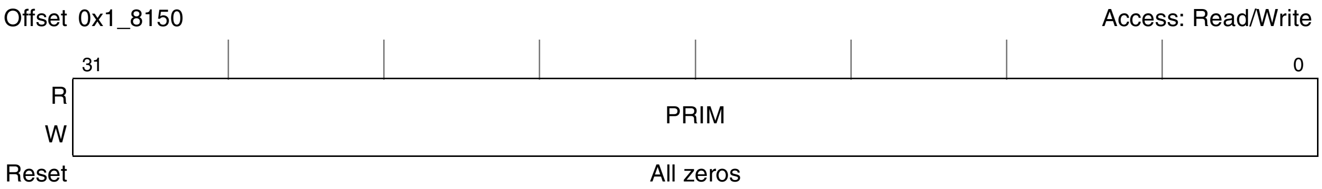


Figure 19-21. Link Layer Configuration Register1 (LinkCfg1)

Table 19-21 describes the LinkCfg2 fields.

Table 19-21. LinkCfg2 Field Descriptions

| Bit | Name | Description |
|---|---|---|
| 31–0 | PRIM | This 32-bit bus specifies the data to be used in the overriding primitive debug logic, described in the definition of LinkCfg1 register. |

## 19.3.4.6 Link Layer Status Register (LinkStatus)

LinkStatus, shown in Figure 19-22, indicates the status of the link layer.

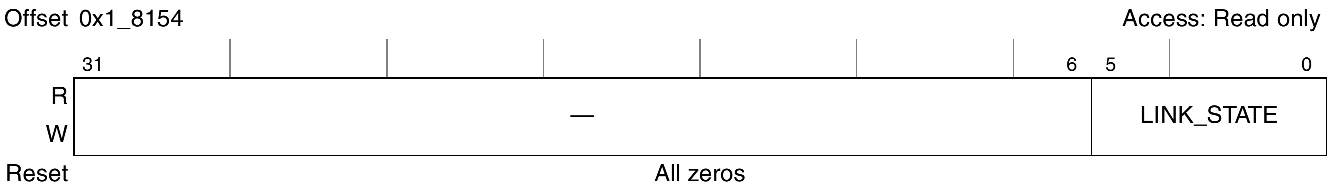


Figure 19-22. Link Layer Status Register (LinkStatus)

Table 19-22 describes the LinkStatus fields.

**Table 19-22. LinkStatus Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 31–6 | — | Reserved |
| 5–0 | LINK_STATE | Current value of the link layer state machine at the time the LinkStatus register is read.<br>L_Reset = 0<br>L_Idle = 1<br>HL_SendChkRdy = 2<br>DL_SendChkRdy = 3<br>L_TPMPartial = 4<br>L_TPMSlumber = 5<br>L_RcvWaitFifo = 6<br>L_PMOff = 7<br>L_PMDeny = 8<br>L_NoCommErr= 9<br>L_NoComm = 10<br>L_SendAlign = 11<br>L_SendSOF = 12<br>L_SendData = 13<br>WAIT_FOR_SYNC = 14<br>L_SendCRC = 15<br>L_SendHold = 16<br>L_RcvHold = 17<br>L_SendEOF = 18<br>L_Wait = 19<br>L_ChkPhyRdy = 20<br>BIST1 = 42<br>L_NoCommPower = 21<br>L_WakeUp1 = 22<br>L_WakeUp2 = 23<br>L_RcvChkRdy = 24<br>L_RcvData = 25<br>L_BadEnd = 26<br>L_RcvEOF = 27<br>L_SendHoldA = 28<br>L_Hold = 29<br>L_GoodCRC = 30<br>L_GoodEnd = 31<br>L_PMOff_2 = 32<br>L_PMOff_3 = 33<br>L_PMOff_4 = 34<br>WAIT_PMACK_SENT_1 = 35<br>WAIT_PMACK_SENT_2 = 36<br>WAIT_PMACK_SENT_3 = 37<br>WAIT_PMACK_SENT_4 = 38<br>WAIT_PMACK_SENT_5 = 39<br>WAIT_PMACK_SENT_6 = 40<br>BIST0 = 41 |

### 19.3.4.7 Link Layer Status Register1 (LinkStatus1)

LinkStatus1, shown in Figure 19-23, indicates the status of the link layer.

Offset 0x1_8158 Access: Read only

| | 31 24 | 23 16 | 15 8 | 7 0 |
|---|---|---|---|---|
| R<br>W | KEGC | PIEGC | CEGC | DEGC |
| Reset | All zeros | | | |

**Figure 19-23. Link Layer Status Register1 (LinkStatus1)**

Table 19-23 describes the LinkStatus1 fields.

**Table 19-23. LinkStatus1 Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 31–24 | KEGC | Kchar error gray count. The number of words received from the PHY, where one or more control character errors have been detected. A value of 255 indicates an error count of 255 or more as this counter does not wrap around to 0. The count value is updated with its current value each time the Status1 register is read. The count is represented in gray code. |
| 23–16 | PIEGC | PHY internal error gray count. The number of words received from the PHY, where one or more internal errors have been detected. A value of 255 indicates an error count of 255 or more as this counter does not wrap around to 0. The count value is updated with its current value each time the Status1 register is read. The count is represented in gray code. |
| 15–8 | CEGC | Code error gray count: The number of words received from the PHY, where one or more code errors have been detected. A value of 255 indicates an error count of 255 or more as this counter does not wrap around to 0. The count value is updated with its current value each time the Status1 register is read. The count is represented in gray code. |
| 7–0 | DEGC | Disparity error gray count. The number of words received from the PHY, where one or more disparity errors have been detected. A value of 255 indicates an error count of 255 or more as this counter does not wrap around to 0. The count value is updated with its current value each time the Status1 register is read. The count is represented in gray code. |

Sample C code to convert gray counts to binary:

```
int Gray2Binary(int gray)
{
int ish;
unsigned long ans,idiv;
ish=1; This is the more complicated direction: In hierarchical stages, starting with a one-bit
right shift, cause each bit to be XORed with all more significant bits.
Ans=gray;
for (;;)
    {
    ans ^= (idiv=ans >> ish);
    if (idiv <= 1 || ish == 16) return ans;
    ish <<= 1; Double the amount of shift on the next cycle.
    }
}
```

### 19.3.4.8 PHY Control Configuration Register1 (PhyCtrlCfg1)

PhyCtrlCfg1, shown in Figure 19-24, controls the configuration of the link layer.

Offset 0x1_815C Access: Read/Write

| | 31–16 |
|---|---|
| R | — |
| W | |
| Reset | All zeros |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | — | | | ENDEC_ EN[1] | — | | | FPRFTI | | — | | | LPB_EN | | | — |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

[1] Reset value must be preserved when writing to the register.

**Figure 19-24. PHY Control Configuration Register1 (PhyCtrlCfg1)**

Table 19-24 describes the PhyCtrlCfg1 fields.

**Table 19-24. PhyCtrlCfg1 Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 31–13 | — | Reserved |
| 12 | ENDEC_EN | Encode decode enable. When asserted high, it enables the PCS to operate in 8 bit mode, and to enable 8B/10B encoding and decoding. When negated low, the PCS is configured to operate in 10-bit mode; the 8B/10B encoder/decoders are bypassed and it is assumed this is done elsewhere. |
| 11–9 | — | Reserved. Reset value must be preserved when writing to the register. |
| 8–7 | FPRFTI | Force PHY ready, force Tx idle. This pair of signals determines how phyRdy is driven, how the output buffer IDLE condition is controlled and how disparity errors in ALIGN primitives should be tolerated during OOB. The IDLE condition is defined in SATA as both traces of the transmit differential pair being driven to common mode.<br>• frcPhyRdy = 0<br>• frcTxIdle = 0<br>In this mode phyRdy and Tx buffer IDLE control driven by OOB state machine. Disparity errors in ALIGN primitives are not tolerated during OOB.<br>• frcPhyRdy = 0<br>• frcTxIdle = 1<br>In this mode phyRdy and Tx buffer IDLE control driven by OOB state machine.<br>Disparity errors in ALIGN primitives are tolerated during OOB.<br>• frcPhyRdy = 1<br>• frcTxIdle = 0<br>In this mode phyRdy is asserted high and Tx buffer IDLE control is forced off, causing the output buffer to be enabled. Tolerance of disparity errors in ALIGN primitives is of no consequence, because OOB is bypassed.<br>• frcPhyRdy = 1<br>• frcTxIdle = 1<br>In this mode phyRdy is asserted high and Tx buffer IDLE control is forced on, causing the output buffer to the IDLE condition. Tolerance of disparity errors in ALIGN primitives is of no consequence as OOB is bypassed. |
| 6–4 | — | Reserved |

Table 19-24. PhyCtrlCfg1 Field Descriptions (continued)

| Bit | Name | Description |
|---|---|---|
| 3–1 | LPB_EN | Loopback enable. These bits control both loopback modes and power management modes.<br>000 No loopback and in normal power mode<br>001 Far end re-timed (parallel) loopback enabled<br>010 Near end analog (serial) loopback enabled<br>011 Invalid<br>100 Invalid<br>101 goPartial. This encoding results in the OOB state machine entering the partial state. Note that in the PCS, partial and slumber have the same effect.<br>110 goSlumber. This encoding results in the OOB state machine entering the slumber state. Note that in the PCS, partial and slumber have the same effect.<br>111Invalid<br>**Note:** This field is available only for SATA1. |
| 0 | — | Reserved |

### 19.3.4.9 Link Layer Command Status Register (CommandStatus)

CommandStatus, shown in Figure 19-25, indicates the status of the command layer.

Offset 0x1_8160 Access: Read only

| | 31 24 | 23 20 | 19 16 |
|---|---|---|---|
| R | — | DMATCmdNo | DMATPMPNo |
| W | | | |
| Reset | All zeros | | |

| | 15 12 | 11 8 | 7 0 |
|---|---|---|---|
| R | RecPMPLocal | CSState | CMState |
| W | | | |
| Reset | All zeros | | |

Figure 19-25. Link Layer Command Status Register (CommandStatus)

Table 19-25 describes the CommandStatus fields.

Table 19-25. CommandStatus Field Descriptions

| Bit | Name | Description |
|---|---|---|
| 31–24 | — | Reserved |
| 23–20 | DMATCmdNo | |
| 19–16 | DMATPMPNo | |
| 15–12 | RecPMPLocal | |
| 11–8 | CSState | CSSIdle = 0x0<br>CSSGP = 0x1<br>CSSPCA = 0x2<br>CSSNPS = 0x4<br>CSSSC = 0x5<br>CSSNCS = 0x6<br>CSSPF = 0x7<br>CSSTO = 0x8<br>CSSWCI = 0x9 |

**Table 19-25. CommandStatus Field Descriptions (continued)**

| Bit | Name | Description |
|---|---|---|
| 7–0 | CMState | CMIdle = 0x00<br>CMFatalError = 0x01<br>CMWE = 0x02<br>CMRF = 0x03<br>CMSNDF = 0x04<br>CMWSNDF = 0x05<br>CMWSNDFWUCA = 0x06<br>CMCIWNE = 0x07<br>CMCIWIF = 0x08<br>CMCISNDS = 0x09<br>CMCIWDMAC = 0x0A<br>CMRUF = 0x0B<br>CMRUFUS = 0x0C<br>CMRUFWUS = 0x0D<br>CMRWSU = 0x0E<br>CMRUFWMW = 0x0F<br>CMSDB = 0x10<br>CMSDBWSN = 0x11<br>CMSDBCleanACK = 0x12<br>CMSDBCCT = 0x13<br>CMSDBPRT = 0x14<br>CMSDBNT = 0x15<br>CMSDBLT = 0x16<br>CMSDBFCC = 0x17<br>CMSDBNCC = 0x18<br>CMSDBLCC = 0x19<br>CMSDBWFT = 0x1A<br>CMRDMAS = 0x1B<br>CMRDMASTC = 0x1C<br>CMRDMASTNC = 0x1D<br>CMRDMASTLC = 0x1E<br>CMRDMASTT = 0x1F<br>CMRDMASTL = 0x20<br>CMRDMASWFTF = 0x21<br>CMRDMASDW = 0x22<br>CMPIOS = 0x23<br>CMPIOSWFTF = 0x24<br>CMPIOSDW = 0x25<br>CMSATAPI = 0x26<br>CMWFC = 0x27<br>CMDF = 0x28<br>CMDFWD = 0x29<br>CMDFWDW = 0x2A<br>CMDFWCRC = 0x2B<br>CMDFBSY = 0x2C<br>CMDMAA = 0x2D<br>CMDMAAWFTF = 0x2E<br>CMDMAADW = 0x2F<br>CMSD = 0x30<br>CMDC = 0x31<br>CMWDC = 0x32<br>CMWU = 0x33<br>CMWRFD = 0x34<br>CMWCC = 0x35<br>CMRUNF = 0x36<br>CMRUNFC = 0x37<br>CMFatalErrorUpdate = 0x38 |

## 19.3.5 System Control Registers

### 19.3.5.1 System Priority Register (SYSPR)

SYSPR, shown in Figure 19-26, can be used to control various settings that affect the system response to DMA operations. Note that the bit ordering is 0–31, rather than 31–0 of the other registers in this chapter.

Offset 0x1_8410

Access: Read/Write

|  | 0–15 |
|---|---|
| R<br>W | — |
| Reset | All zeros |

|  | 16 | 17 | 18 | 19 | 20–28 | 29 | 30–31 |
|---|---|---|---|---|---|---|---|
| R<br>W | PRI_DATA | — | PRI_DES | — | — | RD_SAFE | — |
| Reset | All zeros |  |  |  |  |  |  |

**Figure 19-26. System Priority Register (SYSPR)**

Table 19-26 describes the SYSPR fields.

**Table 19-26. SYSPR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–15 | — | Reserved |
| 16 | PRI_DATA | Data high priority enable.<br>0 Normal operation. SATA data transfers have default (low) priority.<br>1 High priority enabled for SATA data transfers. |
| 17 | — | Reserved |
| 18 | PRI_DES | Descriptor fetch high priority enable.<br>0 Normal operation. SATA descriptor fetches have default (low) priority.<br>1 High priority enabled for SATA descriptor fetches. |
| 19 | — | Reserved |
| 10–28 | — | Reserved |
| 29 | RD_SAFE | Read safe. This bit should be set only if the target of the read DMA operation is a well behaved memory that is not affected by the read operation and which will return the same data if read again from the same location. This means that unaligned reading operation can be rounded up to enable more efficient read operations.<br>0 It is not safe to read more bytes that were intended.<br>1 It is safe to read more bytes that were intended. |

## 19.3.6 Command Header

Each entry in the command header table consists of the structure shown in Figure 19-27.

**NOTE**

In this chapter, “word” refers to 4 bytes or 32 bits.

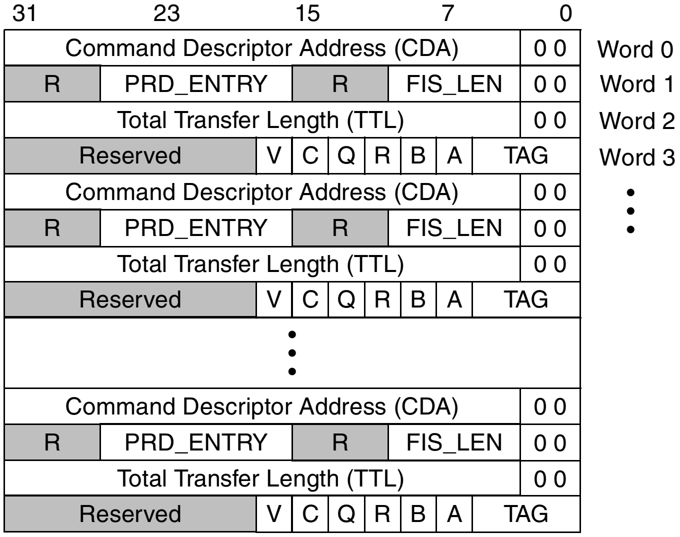


**Figure 19-27. Command Header**

Table 19-27 shows word 0—data base address—of the command header.

Table 19-27. Word 0—Data Base Address

| Bit | Name | Description |
|---|---|---|
| 31–2 | CDA | Command descriptor base address. Indicates the 32-bit physical address of the command descriptor block. The block must be word-aligned, indicated by bits 1–0 being reserved. |
| 1–0 | — | Reserved |

Table 19-28 shows word 1—FIS_LEN—of the command header.

Table 19-28. Word 1—FIS_LEN

| Bit | Name | Description |
|---|---|---|
| 31–22 | — | Reserved |
| 21–16 | PRD_ENTRY | Number of PRD entries including indexed entries. |
| 7 | — | Reserved |
| 6–2 | FIS_LEN | FIS length. This is a 5-bit word count of the total length of the control or vendor-specific FIS to transfer. |
| 1–0 | — | Reserved |

Table 19-29 shows word 2—data base address—of the command header.

Table 19-29. Word 2—Data Base Address

| Bit | Name | Description |
|---|---|---|
| 31–2 | TTL | Total transfer length. This is a 30-bit word count of the total length of the data transfer. It is used to detect overruns/underruns between the transfer lengths programmed in the command and the PRDT. |
| 1–0 | — | Reserved |

Table 19-30 shows word 3—description information—of the command header.

Table 19-30. Word 3—Description Information

| Bit | Name | Description |
|---|---|---|
| 31–12 | — | Reserved |
| 11 | — | Reserved, should be 1. |
| 10 | V | Vendor BIST. When this bit is set, it indicates that the command is a Vendor BIST, thus FIS will loop back at the PHY local test. |
| 9 | C | Snoop enable during all descriptor read/write operations associated with this command. |
| 8 | Q | Queued. Command is an FPDMA queued command. |
| 7 | R | Reset. The command is a SRST or device reset. |
| 6 | B | BIST. The command will require the host to enter BIST mode. |

Table 19-30. Word 3—Description Information (continued)

| Bit | Name | Description |
|---|---|---|
| 5 | A | ATAPI command. The command is an ATAPI command and thus will require that the host uses the ATAPI portion of the command descriptor to issue the command. The CFIS also has to be written with the packet command. |
| 4–0 | TAG | The 5-bit TAG assigned by software for command tracking. It is the same as the value written to the command register host-to-device. |

## 19.3.7 Command Descriptor

As shown in Figure 19-28, each entry in the command list points to a structure called the command descriptor.

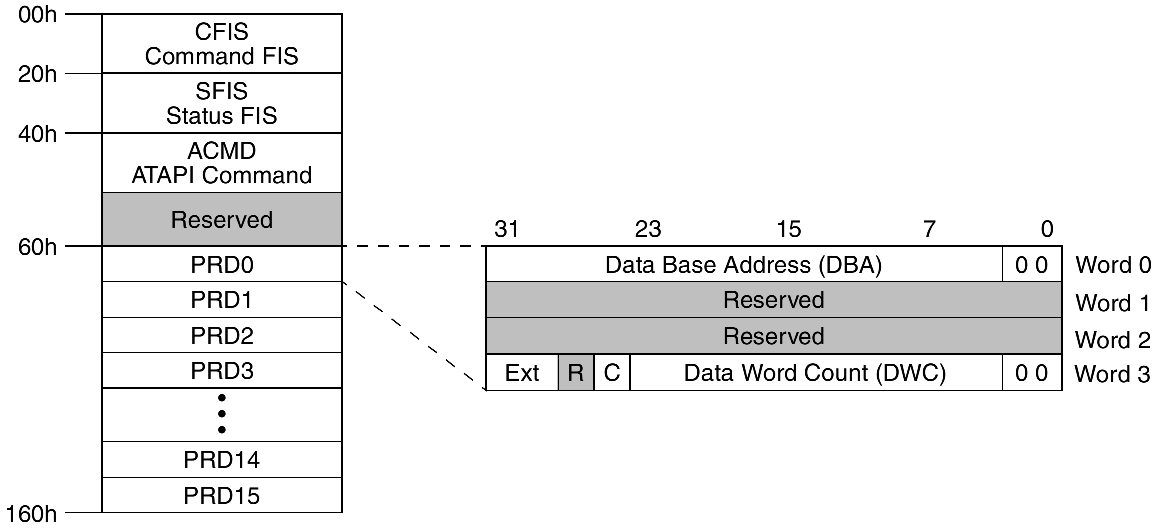


Figure 19-28. Command Descriptor

### 19.3.7.1 Command FIS Non-Queued Commands (CFIS)

Command FIS is a software constructed FIS. For data transfer operations, this is the H2D Register FIS format as specified in the Serial ATA 2.5 standard. The SATA controller fetches this from memory and sends the appropriate amount of data to the attached port. If a port multiplier is attached, this field must have the port multiplier port number in the FIS itself. CFIS lengths are two to eight words and must be in word granularity. A typical command FIS is shown in Figure 19-29.

Register FIS host-to-device

|  | 31 24 | 23 16 | 15 12 | 11 8 | 7 0 |
|---|---|---|---|---|---|
| 0 | Features | Command | C R R R | PM Port | FIS Type (27h) |
| 1 | Device | LBA High | LBA Mid |  | LBA Low |
| 2 | Features (exp) | LBA High (exp) | LBA Mid (exp) |  | LBA Low (exp) (0) |
| 3 | Control | Reserved (0) | Sector Count (exp) |  | Sector Count |
| 4 | Reserved (0) | Reserved (0) | Reserved (0) |  | Reserved (0) |

**Figure 19-29. Register Host-to-Device**

### 19.3.7.2 Command FIS First Party DMA Commands NCQ

Figure 19-30 shows register host-to-device first party DMA commands NCQ. The shaded components show where this FIS differs for the non-NCQ register host-to-device FIS.

Register FIS host to device—Read/write FPDMA queued

|  | 31 24 | 23 16 | 15 12 | 11 8 | 7 3 | 2 0 |
|---|---|---|---|---|---|---|
| 0 | Features<br>Sector Count 7:0 | Command | C R R R | PM Port | FIS Type (27h) |  |
| 1 | Device | LBA High | LBA Mid |  | LBA Low |  |
| 2 | Features (exp)<br>Sector Count 15:8 | LBA High (exp) | LBA Mid (exp) |  | LBA Low (exp) (0) |  |
| 3 | Control | Reserved (0) | Sector Count (exp)<br>Reserved (0) |  | Sector Count<br>TAG |  |
| 4 | Reserved (0) | Reserved (0) | Reserved (0) |  | Reserved (0) |  |

**Figure 19-30. Register Host-to-Device First Party DMA Commands NCQ**

### 19.3.7.3 Status FIS (SFIS)

This FIS is created in hardware. For normal operations, this is the D2H register FIS format as specified in the Serial ATA 2.5 standard. SFIS lengths are two to eight words and must be of word granularity. A typical status FIS is shown in Figure 19-31.

Register FIS device to host

|  | 31 24 | 23 16 | 15 12 | 11 8 | 7 0 |
|---|---|---|---|---|---|
| 0 | Error | Status | R I R R | PM Port | FIS Type (34h) |
| 1 | Device | LBA High | LBA Mid |  | LBA Low |
| 2 | Reserved (0) | LBA High (exp) | LBA Mid (exp) |  | LBA Low (exp) (0) |
| 3 | Reserved (0) | Reserved (0) | Sector Count (exp) |  | Sector Count |
| 4 | Reserved (0) | Reserved (0) | Reserved (0) |  | Reserved (0) |

**Figure 19-31. Register Device-to-Host**

### 19.3.7.4 ATAPI Command (ACMD)

This is a software constructed region of three or four words in length that contains the ATAPI command to transmit if the ‘A’ bit is set in the command header. The ATAPI command must be either 12 or 16 bytes in length. The length transmitted by the SATA IP is determined by the PIO setup FIS that is sent by the device requesting the ATAPI command.

### 19.3.7.5 Physical Region Descriptor Table (PRDT)

This is a software constructed table of addresses to use to complete the data transfer. Up to 16 structures can be supported in the current command descriptor. The format of the address entry is defined by the "Block vector structures for passing segmented data type of the IEEE Std. 1212.1-1993". The total definable length supported in the 16 entries is 64 Mbytes.

Table 19-31 shows word 0—data base address—of the PRDT.

**Table 19-31. Word 0—Data Base Address**

| Bit | Name | Description |
|---|---|---|
| 31–2 | DBA | Data base address. Indicates the 32-bit physical address of the data block. The block must be word aligned, indicated by bits 1–0 being "reserved, must be 00." |
| 1–0 | — | Reserved |

Table 19-32 shows word 3—description information—of the PRDT.

**Table 19-32. Word 3—Description Information**

| Bit | Name | Description |
|---|---|---|
| 31 | EXT | If the extension flag is set to 1, then the DBA field contains the address of the extension segment, and the DWC field contains the size of this extension segment (this is called an "indirect descriptor"). |
| 30–23 | — | Reserved |
| 22 | C | Data snoop enable bit. When this bit is set, all data read/write operations associated with the PRD entry for this command will be snoopable. |
| 21–2 | DDC | Data word count. A 0-based value that Indicates the length, in words, of the data block. A maximum length of 4 Mbytes may exist for any entry. Bits 1–0 of this field must always be 0 to indicate that size is in words. A value of 0x0_0000 indicates a full 4 Mbytes transfer. |
| 1–0 | — | Reserved |

## 19.3.8 Vendor-Specific BIST Operation

As part of the host self-diagnostic operation, a vendor-specific BIST mode is supported. This mode, in conjunction with a PHY that supports serial loopback, allows for the test of the SATA controller operation.

The mode exercises the following paths:

- DMA controller FIS transmission
- Command layer FIS transmission
- Transport layer Tx FIFO FIS transmission
- Link layer FIS transmission
- PHY modes
- Link layer FIS reception

- Transport layer Rx FIFO and FIS reception
- Command layer FIS reception
- Command layer FIS reception
- Host DMA controller FIS reception

To run this self-test on SATA1, lane A, the software performs the following operations:

**Table 19-33. Vendor BIST Test—Command Header**

| Word Number | Hexadecimal Value |
|---|---|
| Word 0 | CDA |
| Word 1 | 0x0000_000C |
| Word 2 | 0x0000_0000 |
| Word 3 | 0x0000_0400 |

**Table 19-34. Vendor BIST Test—Command Descriptor**

| Word Number | Hexadecimal Value | Comments |
|---|---|---|
| Word 0 | 0x0001_0058 | — |
| Word 1 | 0xAAAA_A034 | AAAA_A is the first test pattern (can be any value) |
| Word 2 | 0xBBBB_B034 | BBBB_B is the second test pattern (can be any value) |
| Word 3 | Reserved | Reserved, must be all zeros |
| Word 4 | Reserved | Reserved, must be all zeros |
| Word 5 | Reserved | Reserved, must be all zeros |
| Word 6 | Reserved | Reserved, must be all zeros |
| Word 7 | Reserved | Reserved, must be all zeros |

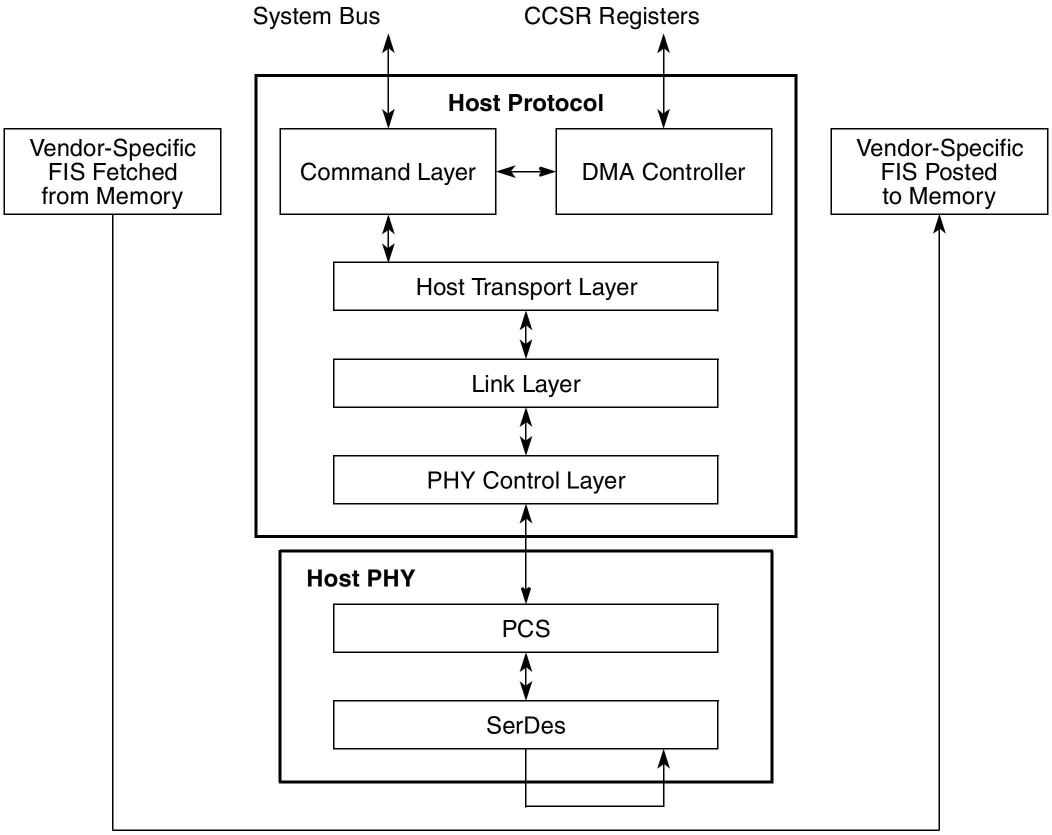


**Figure 19-32. Vendor-Specific BIST Operation**

## 19.4 Transport Layer Architectural Overview

The function of the SATA transport layer is to interface between the command and link layers in the transmission and reception of FIS.

On the transmit path, the transport layer frames the FIS's placed into the Tx FIFO. The FIS's are framed based on a programmed length for non-data FIS and are a configurable length for data FIS. When the transport layer is instructed to send a non-data FIS, it employs a retry policy until the far end signals acceptance of the transmitted FIS.

On the reception path, the transport layer deframes the FIS's and places them into the Rx FIFO. When an FIS is received, the transport layer informs the command layer. For a non-data FIS, the FIS is considered received when the end-of-frame (EOF) is signaled by the link layer and the FIS has been received with a good CRC. For a short vendor-specific FIS, the FIS is considered as a non-data FIS. For a longer vendor-specific FIS, the FIS reception is signaled when the RX FIFO reaches its water mark. For a data FIS, the FIS is considered received when the first word (header) is written into the FIFO.

The receive FIFO is written with data contained in the FIS sent by the link layer. When the data is stable at the output of the receive FIFO, the command layer can take the data. If the command layer is not ready to accept the data, the data builds up in the receive FIFO. When the receive FIFO exceeds its threshold, the transport layer stalls the link layer, which will in turn send HOLD primitives to the far end. This threshold takes into consideration the latency involved in getting the far end to stop transmitting the data. This threshold is programmable to allow for the use of high-latency repeaters or retainers in between the host and device.

The transmit FIFO is written with data to be sent in the FIS transferred by the DMA controller. When the data is stable at the output of the transmit FIFO, the link layer can take the data. If the transmit FIFO cannot supply data to the link layer, the transport layer stalls the link layer, which will in turn send HOLD primitives to the far end.

## 19.5 Link Layer Overview

The function of the SATA link layer is to interface between the transport and physical layers in the transmission and reception of frames and primitives. The link layer utilizes the two unidirectional links provided by the SATA interface to maintain coordinated communication between the host and the device. Payload data can only be transmitted in one direction at a time. The link layer can work at either SATA first-generation (1.5 Gbps) or second-generation 2 (3 Gbps) speeds.

On transmit, the link layer first communicates with the peer far end link layer to determine if it is ready to receive. Assuming the far end link layer can receive data, the local link layer can then begin to take data in the form of words from its transport layer. It inserts start-of-frame (SOF) before the start of the data portion of a frame, calculates and inserts the CRC after the data portion of a frame, and inserts the EOF primitive at the end. The link layer scrambles the contents of the frame, including the calculated CRC, but excluding the SOF and EOF diameters and any other embedded primitives. The 8B/10B encoding of the data is done in the PHY layer. At the end of the transmission, the link layer reports transmission status to the transport layer.

On receive, the link layer first acknowledges its readiness to receive with its peer link layer. Then it awaits reception of the SOF primitive that marks the start of the received data. Following detection of the SOF primitive, the link layer proceeds to accept the incoming data. The 8B/10B decoding of the data is done in the PHY layer. Next, the link layer removes all primitives including the SOF and EOF diameters. It then descrambles the contents of the frame. The link layer also calculates the CRC on the incoming frame between the SOF and EOF delimiters, and compares this calculated value to the received value. Any mismatch is reported to the transport layer. During frame reception, disparity or code errors are reported to the command layer, and appropriate action is taken in the link layer. The descrambled and decoded receive data stream is passed to the transport layer as the frame is being received. Finally, at the end of the frame, the link layer reports reception status to the transport layer.

The link layer also partakes in flow control between the local and remote ends. The layer supports flow control actions based on the local FIFO status (located in the transport layer), or in response to receiving flow control messages from the remote end.

The transmit side of the link layer is also responsible for inserting a pair of ALIGN primitives every 254 words, or more frequently if programmed by the user.

### 19.5.1 Link Layer functionality

The link layer is composed of a number of functions:

- Link layer state machines
- Frame content scrambler and descrambler
- CRC generation and checking
- Bus interfaces to PHY and transport layer
- CONT primitive processing
- ALIGN insertion on transmit
- Debug functionality
- BIST support
- Link layer state machines

The four link layer state machines are described in the following sections.

#### 19.5.1.1 Link Idle State Machine

The link idle state machine is responsible for detecting a transmit request from the transport layer or a frame reception request from the far end. The state machine arbitrates whether these two events coincide. The SATA specification defines that the host end always backs down in this case. Furthermore, this machine interprets power mode change requests from both the transport and PhyCtrl layers and initiates actions to enable the power mode change. Power mode change can only occur if the feature is enabled via the PhyCtrlCfg register LPB_EN bits. Finally, this state machine also detects the negation and assertion of PHY_READY from the PHY and notifies the transport layer of the change.

#### 19.5.1.2 Transmit State Machine

This state machine is responsible for frame transmission to the PHY. The state machine places the SOF and EOF headers on each frame, calculates the CRC, and inserts it before the EOF delimiter. Between the SOF and CRC markers, the link layer accepts the current word from the transport layer and uses this as the next word of the frame. The link layer also inserts a pair of ALIGN primitives every 254 words of frame data. Finally, at the end of the frame transmission, the state machine waits for status from the far end link layer via received R_OK or R_ERR primitives. If the far end received the frame correctly, the local link layer signals TX_OK to the transport layer; otherwise, it signals TX_NOT_OK to the transport layer.

The transmit state machine also partakes in flow control actions, if necessary, during packet transmission. If the transport layer cannot supply a new word and the frame is not finished, the transmit state machine responds by sending HOLD primitives until the transport layer is ready with valid frame data. Also, during frame transmission, if the state machine detects a received HOLD primitive from the PHY layer, it interrupts the current frame transmission and sends HOLDA primitives to the PHY to be transmitted to the far end.

The current frame transmission can only be aborted by two events. The first is on reception of a DMAT primitive from the far end. In this case, the link layer state machine stops the current transfer and calculates and inserts the current CRC. This is a controlled termination. The second is when the transport layer wishes to send a control register frame signaled via TRANSMIT_CRF.

If at any point in the frame transmission process, the link layer detects error conditions, it signals these to the command layer. The errors can occur if the link layer detects the following conditions:

- PHY_READY negates
- SYNC primitive is received during frame transmission

### 19.5.1.3 Receive State Machine

This state machine is responsible for frame reception from the PHY layer. The state machine removes the SOF and EOF headers and other primitives from each frame, calculates the CRC, and compares it to the received CRC. Between the SOF and CRC markers, the link layer accepts the current word from the Phy layer and uses this as the next word of the frame, transferring it to the transport layer. At the end of the frame reception, if the calculated CRC is not the same as the received CRC, the link layer signals an error to the transport layer. This is done via RX_CRC_OK and RX_CRC_NOT_OK. During frame reception, if no errors are detected, the link layer transmits R_IP primitives to the far end peer link layer. Finally, at the end of the frame reception, the link layer sends the R_OK primitive if no error was detected during reception. If an error was detected, it sends a R_ERR primitive instead.

The receive state machine also partakes in flow control actions if necessary, during FIS reception. If the transport layer cannot accept a new word, (because its receive FIFO has reached its watermark level), and the FIS is not finished, the receive state machine responds by sending HOLD primitives on the back channel until such time as the transport layer is ready to accept FIS data again. Also, during FIS reception, if the state machine detects a received HOLD primitive from the far end, it responds by sending HOLDA primitives to the far end.

The current frame reception can be interrupted if the transport layer wishes to send a control register frame, signaled via TRANSMIT_CRF.

If at any point in the frame reception process, the link layer detects error conditions, it signals these to the command layer. The errors can occur if the link layer detects the following conditions:

- PHY_READY negates
- SYNC primitives is received during frame transmission
- WTRM primitive is received before EOF

### 19.5.1.4 Power Mode Change State Machine

This state machine is responsible for handling change of power mode requests. These requests can come from the command layer superset registers or the far end. This state machine responds by transmitting PMREQ_P/PMREQ_S primitives to the far end and waiting for PMACK primitives from it in response. Once PMACK is received, the state machine instructs the PHY layer to enter either a partial or slumber state.

A write to the SControl register SPM field or reception of a COMWAKE from the far end will initiate a resume to active power mode.

If the link layer receives a PMREQ_P/PMREQ_S primitive from a peer link layer and is enabled to perform power management modes (SControl IPM bits are cleared), it responds by sending at least four PMACK primitives. A write to the SControl register SPM field or reception of a COMWAKE from the far end will initiate a resume to active power mode.

If the link layer receives an XRDY primitive from the far end while it is in the partial or slumber state, it returns to idle and signals a link sequence error to the command layer, that is, SError[S] = 1.

### 19.5.1.5 Frame Content Scrambler and Descrambler

There are two separate scramblers used in the SATA controller, one for the data payload and the other for repeated primitive suppression. The contents of each word of data (excluding all primitives) between SOF and EOF must be scrambled before 8B/10B encoding. Scrambling is performed on word quantities according to the following polynomial:

G(X) = X16 + X15 + X13 + X4 + 1

The scrambler is initialized with a seed value of 0xFFFF at each SOF transmission and rolls over every 2048 words. Payload data is scrambled prior to transmission, by XORing the data to be transmitted with the output of this scrambler.

If a CONT primitive is transmitted, then the intervening data between the last CONT primitive and a subsequent primitive must be scrambled also. This scrambler uses the same polynomial as defined above for data payload scrambling and is reset to the initial value upon detection of a COMINIT or COMRESET event. If a CONT primitive is transmitted or received during a frame transfer, then the current data payload scrambler value at the last word is held.

When payload data is received by the link layer it is descrambled by XORing it with the output of its descrambler. The descrambler is re-seeded at the beginning of the received data payload, that is, at each SOF reception. The descrambler uses the same polynomial as the scrambler.

### 19.5.1.6 CRC Generator and Checker

A 32-bit CRC is calculated on the data contents of each frame and is inserted in the word before the EOF. The CRC covers all data bytes in the frame excluding any primitives such as SOF, EOF, HOLD, HOLDA, DMAT, SYNC, X_RDY, R_RDY, or ALIGNs.

The CRC generator works on word quantities. Any padding to the boundary is done in the transport layer. The polynomial used for the CRC is as follows:

G(X) = X32 + X26 + X23 + X22 + X16 + X12 + X11 + X10 + X8 + X7 + X5 + X4 + X2 + X + 1

The CRC is initialized with a seed value of 0x52325032 at each SOF.

The CRC generation or checking does not apply to primitives (as stated above) or to CONT'ed primitives. If a CONT primitive is transmitted or received, then the intervening data between the last CONT primitive and a subsequent primitive is not included in the CRC calculation for a frame. If this happens during a frame transfer, then the current CRC scrambler value at the last word is held.

### 19.5.1.7 8B/10B Encode and Decode

All data and primitives must be encoded prior to transmission on the line. The 8B/10B encode/decode occur in the PHY layer.

### 19.5.1.8 CONT Primitive Processing

Using the CONT primitive, the link layer is capable of replacing repetitive primitive streams with scrambled data. This reduces EMI emissions because primitives are not scrambled.

The link layer can transmit a CONT primitive at a point where it knows it must transmit a number of repeating primitives. After a CONT primitive has been transmitted, the link layer then transmits scrambled junk data to the PHY layer. The content of this junk data is disregarded. At the far end link layer, the reception of a CONT primitive will cause the last received valid primitive to be implied to be repeated until it receives the next valid non-ALIGN primitive. Transmission of a new valid primitive halts the current CONT processing; reception of a new valid non-ALIGN primitive halts the current CONT processing.

This action can occur on transmit and receive. The link layer supports both the transmission and reception of CONT primitives.

### 19.5.1.9 ALIGN Insertion

The link layer is responsible for ALIGN insertion and removal at a fixed frequency. A pair of ALIGN primitives are inserted into the transmit data stream every 254 words. At the receive end, the ALIGN primitives are stripped from the incoming data stream in the link layer.

For diagnostic purposes, the rate of ALIGNs can be increased as much as two ALIGNS per one word; for example, ALIGN, ALIGN, data, ALIGN, ALIGN. In addition, the SEND_4_ALIGNS bit can be set to instruct the link layer to send four ALIGNs at a time instead of two.

### 19.5.1.10 Debug Functionality

There are a number of useful features designed into the link layer to aid debug, as follows:

- The align insertion rate can be increased using the ALIGN_RATE register field in the command layer.
- Four error counters can be monitored by issuing register reads to the command layer: the disparity error counter, the code error counter, the PHY internal error counter, and the control character error counter.
- A number of configuration bits in the command layer can be used to override normal primitive insertion. For example,
  - Set PRIM_OVR_STATE = (L_SendHold state (16))
  - Set PRIM  = 0xb5b5957c, that is, a SYNC primitive
  - During the transfer, set PRIM_OVRD_EN = 1
- When the link layer detects a rising edge on PRIM_OVRD_EN, it will insert one SYNC primitive into the datastream in place of the HOLD, when the LINK_STATE reaches the L_SendHold state. Only one HOLD primitive will be overridden; the PRIM_OVRD_EN must be cleared and written to again to force another override to occur.

#### 19.5.1.11 BIST Support

The transmit and receive subblocks of the link layer contain logic to support BIST activate FIS functionality.

When a BIST activate FIS is either received or transmitted successfully by the transport layer, it issues a request to the link layer to enter BIST mode. This forces the link layer to enter a BIST state in its state machine as soon as it receives a SYNC primitive from the far end. In the BIST state, the link layer transmits a data sequence as specified by the two BIST data patterns in the BIST activate FIS. The link layer also monitors the incoming data from the PHY to detect the BIST data pattern is as specified in the BIST activate FIS. When it detects the correct data sequence, the HStatus[BIST_Err] is deasserted. The BIST_Err bit will stay deasserted unless an error occurs in the datastream from the far end.

## 19.6 PHY Control Layer Overview

The PHY control layer operates between the PHY and link layers. On receive, the PHY control layer converts the 16-bit parallel data from the PHY to a 32-bit word, which it presents to the link layer. The PHY control layer aligns the control word of the SATA primitive to the lowest word position of the word. The PHY control layer takes in the per-byte error signals and the per-byte control/data bits output by the PHY and converts them into 4-bit buses, with each bit of the bus corresponding to a byte in the word.

On transmit, the PHY control layer takes in the 32-bit transmit data from the link layer and converts it to 16 bits of data which it presents to the PHY. The control/data bit from the link layer (which is always assumed to be associated with the lowest byte position of the transmit word) is also passed onto the PHY with the appropriate word.

## 19.7 Initialization/Application Information

### 19.7.1 SATA Controller Initialization Steps

These steps bring the SATA controller online, synchronize the SATA controller with the attached device, and issue typical command for execution.

1. Write HControl[HC_ON] = 1 to bring the SATA controller online.
2. Poll the HStatus[HS_ON] till HS_ON = 1, indicating that the controller is online.
3. Poll the SStatus[DET] till DET = 4’b0011 meaning that the device presence is detected and PHY communication is established. In this state, SStatus[SPD] indicates the negotiated communication speed.
4. To read the device’s signature, poll HStatus[SIG_UPD] till it goes up. Read the signature from the SIG register.
5. Initialize the CHBA register to point to the command header block.
6. Build a command header block in memory. Refer to Section 19.3.6, “Command Header.”
7. Build a command descriptor block in memory. Refer to Section 19.3.7, “Command Descriptor.”
8. Build a number of PRD tables in memory as defined by the PRD_NUM field in the command header. Refer to Section 19.3.7.5, “Physical Region Descriptor Table (PRDT).”

9. Initialize the CQPMP register with the device's PM number. If the port multiplier is not used, clear this field.
10. Poll the CQR[CQ*n*] to determine which command can be issued.
11. After CQ*n* is determined, write 1 to CQR[CQ*n*] to issue this command and start execution.
12. Poll the CCR[CC*n*] till CC*n* goes up, indicating that the command is completed.

The following example presents the structure of descriptors to issue the ReadDMA command:

- Build the command header in memory. See Table 19-35, below.

**Table 19-35. Read DMA Command—Command Header**

| Word Number | Hexadecimal Value | Description |
|---|---|---|
| Word 0 | CDA | Pointer to memory where command descriptor begins |
| Word 1 | 0x0002_0014 | Two PRD tables contain the data, FIS length = 20 bytes |
| Word 2 | 0x0000_0200 | Length of data associated with this command = 0x200 |
| Word 3 | 0x0000_0000 | Tag = 0 |

- Build command descriptor in memory. See Table 19-36, below.

**Table 19-36. Read DMA Command—Command Descriptor**

| Word Number | Hexadecimal Value | Description |
|---|---|---|
| Word 0 | 0x00C8_8027 | Command = Read DMA |
| Word 1 | 0x0000_0010 | LBA = 24'h10 |
| Word 2 | 0x0000_0000 | LBA(exp) = 24'h0 |
| Word 3 | 0x0000_0001 | Sector Count = 1 |
| Word 4 | 0x0000_0000 | Reserved |

- Build two PRD entries in memory. See Table 19-37.

**Table 19-37. Read DMA Command—PRD Entries**

| Word Number | Hexadecimal Value | Description |
|---|---|---|
| Word 0 | PRD1 | Address of first portion of data. |
| Word 1 | 0x0000_0000 | Reserved |
| Word 2 | 0x0000_0000 | Reserved |
| Word 3 | 0x0000_0100 | PRD1 contains 0x100 bytes of data |
| Word 4 | PRD2 | Address of second portion of data. |
| Word 5 | 0x0000_0000 | Reserved |
| Word 6 | 0x0000_0000 | Reserved |
| Word 7 | 0x0000_0100 | PRD2 contains 0x100 bytes of data |

- Fill the PRD1 and PRD2 with user-defined data; see Figure 19-33, below.

|  |  |  |  |  |  |
|---|---|---|---|---|---|
| 31 |  |  | 23 15 7 | 0 |  |
| Data Base Address (DBA) |  |  |  | 0 0 | Word 0 |
| Reserved |  |  |  |  | Word 1 |
| Reserved |  |  |  |  | Word 2 |
| Ext | R | C | Data Word Count (DWC) | 0 0 | Word 3 |

Figure 19-33. PRD Entry

# Chapter 20
# Enhanced Secure Digital Host Controller

## 20.1 Overview

The enhanced secure digital host controller (eSDHC) provides an interface between the host system and these types of memory cards:

- MultiMediaCard (MMC)

  MMC is a universal low-cost data storage and communication medium designed to cover a wide area of applications including mobile video and gaming, which are available from either pre-loaded MMC cards or downloadable from cellular phones, WLAN, or other wireless networks. Old MMC cards are based on a seven-pin serial bus with a single data pin, while the new high-speed MMC communication is based on an advanced 11-pin serial bus designed to operate in a low voltage range.
- Secure digital (SD) card

  The secure digital (SD) card is an evolution of old MMC technology. It is specifically designed to meet the security, capacity, performance, and environment requirements inherent in the emerging audio and video consumer electronic devices. The physical form factor, pin assignments, and data transfer protocol are forward-compatible with the old MMC.

The eSDHC acts as a bridge, passing host bus transactions to SD/MMC cards by sending commands and performing data accesses to or from the cards. It handles the SD/MMC protocol at the transmission level. Figure 20-1 shows connection of the eSDHC.

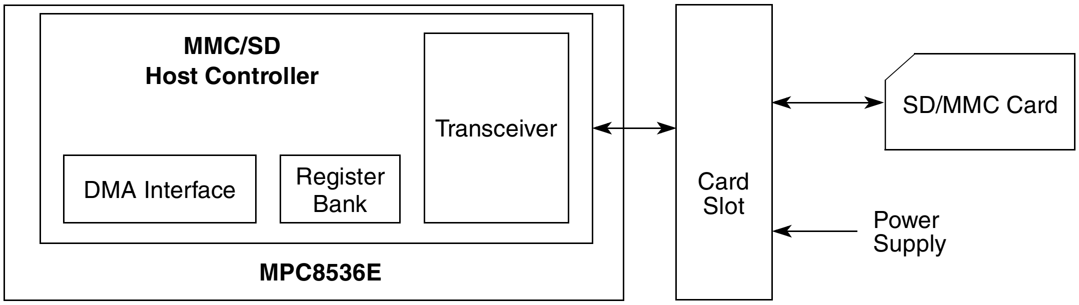


**Figure 20-1. System Connection of the eSDHC**

Figure 20-2 is a block diagram of the eSDHC.

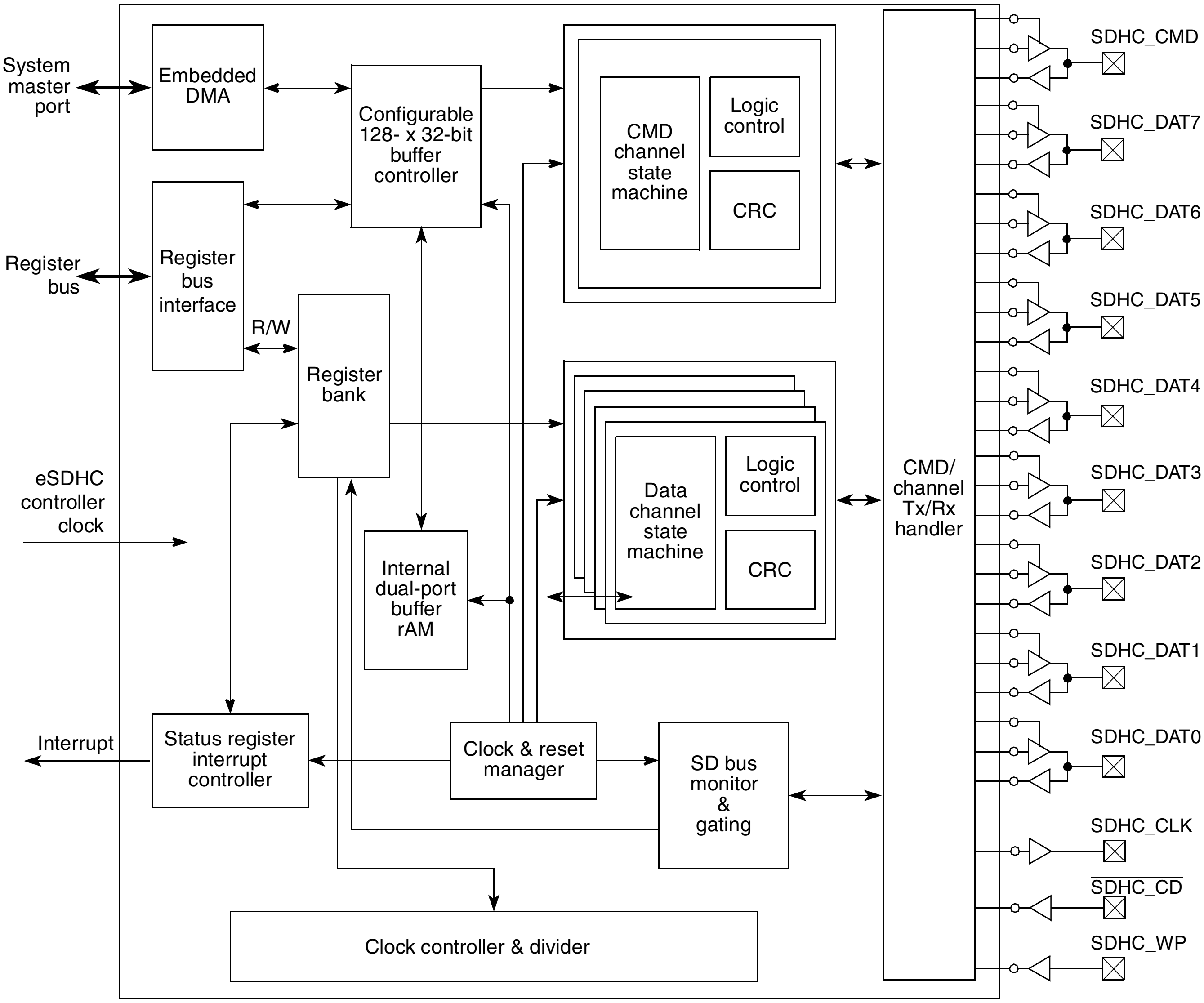


**Figure 20-2. eSDHC Block Diagram**

## 20.2 Features

The eSDHC includes the following features:

- Compatible with the following specifications:
  - *SD Host Controller Standard Specification, Version 2.0* (http://www.sdcard.org) with test event register support
  - *MultiMediaCard System Specification, Version 4.2* (http://www.mmca.org)
  - *SD Memory Card Specification, Version 2.0* (http://www.sdcard.org)

- Designed to work with SD Memory, miniSD Memory, SD Combo, MMC, MMC*plus*, and RS-MMC cards
- Card bus clock frequency up to 52 MHz
- Supports 1-/4-bit SD mode, 1-/4-/8-bit MMC modes
  - — Up to 200 Mbps data transfer for SD/MMC cards using four parallel data lines
  - — Up to 416 Mbps data transfer for MMC using 8 parallel data lines
- Single- and multi-block read and write
- Write-protection switch for write operations
- Synchronous abort
- Pause during the data transfer at a block gap
- Auto CMD12 for multi-block transfer
- Host can initiate non-data transfer commands while the data transfer is in progress
- Fully configurable 128 × 32-bit FIFO for read/write data
- Internal DMA capabilities
- Supports booting from eSDHC. See Section 4.5.1.1, “eSDHC Boot,” for more detailed information

### 20.2.1 Data Transfer Modes

The eSDHC can select the following modes for data transfer:

- SD 1-bit
- SD 4-bit
- MMC 1-bit
- MMC 4-bit
- MMC 8-bit
- Identification mode (up to 400 kHz)
- MMC full-speed mode (up to 20 MHz)
- MMC high-speed mode (up to 52 MHz)
- SD full-speed mode (up to 25 MHz)
- SD high-speed mode (up to 50 MHz)

## 20.3 External Signal Description

The eSDHC has 12 chip I/O signals.

- SDHC_CLK is the internally generated clock signal that drives the MMC, or SD card.
- SDHC_CMD I/O sends commands and receive responses from the card.
- SDHC_DAT7–SDHC_DAT0 perform data transfers between the eSDHC and the card. If the eSDHC is desired to support a 4-bit data transfer, SDHC_DAT7–SDHC_DAT4 can also be optional and tied high.

- $\overline{\text{SDHC\_CD}}$ and SDHC_WP are card detection and write protection signals from the socket.
  — Signals $\overline{\text{SDHC\_CD}}$ and SDHC_WP are optional for system implementation.

Table 20-1 shows the properties of the eSDHC I/O signals.

**Table 20-1. Signal Properties**

| Name | Port | Function | Reset State | Pull up/Pull down Required |
|---|---|---|---|---|
| SDHC_CLK | O | Clock for MMC/SD card | 0 | N/A |
| SDHC_CMD | I/O | Command line to card | High impedance | Pull up |
| SDHC_DAT7 | I/O | **8-bit mode:** DAT7 line<br>not used in other modes | High impedance | Pull up |
| SDHC_DAT6 | I/O | **8-bit mode:** DAT6 line<br>not used in other modes | High impedance | Pull up |
| SDHC_DAT5 | I/O | **8-bit mode:** DAT5 line<br>not used in other modes | High impedance | Pull up |
| SDHC_DAT4 | I/O | **8-bit mode:** DAT4 line in<br>not used in other modes | High impedance | Pull up |
| SDHC_DAT3 | I/O | **4-/8-bit mode:** DAT3 line or configured as card detection pin<br>**1-bit mode:** May be configured as card detection pin | High impedance | Board should have 100K pull down. The card drives 50K pull up as required by the SD card specification. |
| SDHC_DAT2 | I/O | **4-/8-bit mode:** DAT2 line or read wait<br>**1-bit mode:** Read wait | High impedance | Pull up |
| SDHC_DAT1 | I/O | **8-bit mode:** DAT1 line<br>**4-bit mode:** DAT1 line or interrupt detect<br>**1-bit mode:** Interrupt detect | High impedance | Pull up |
| SDHC_DAT0 | I/O | DAT0 line or busy-state detect | High impedance | Pull up |
| $\overline{\text{SDHC\_CD}}$ | I | Card detection pin; if not used, tie high.<br>Low Card present<br>High No card present | N/A | N/A |
| SDHC_WP | I | Card write protect detect; if not used, tie to logic corresponding to write enabled, as shown in Section 23.4.1.8, “General Configuration Register (GENCFGR)”:<br>If GENCFGR[SDHC_WP_INV]=0:<br>Low Write enabled<br>High Write protected<br>If GENCFGR[SDHC_WP_INV]=1:<br>Low Write protected<br>High Write enabled | N/A | N/A |

## 20.4 Memory Map/Register Definition

Table 20-2 shows the memory mapped registers of the eSDHC module and their offsets. It lists the offset, name, and a cross-reference to the complete description of each register. Note that the full register address is comprised of CCSRBAR together with the eSDHC block base address and offset listed in Table 20-2. Undefined 4-byte address spaces within offset 0x000–0xFFF are reserved.

**NOTE**

All eSDHC registers must be accessed as aligned 4-byte quantities. Accesses to the eSDHC registers that are less than 4-bytes are not supported.

**Table 20-2. eSDHC Memory Map**

| eSDHC Registers—Block Base Address 0x2_E000 | | | | |
|---|---|---|---|---|
| **Offset** | **Register** | **Access** | **Reset** | **Section/Page** |
| 0x000 | DMA system address (DSADDR) | R/W | 0x0000_0008 | 20.4.1/20-6 |
| 0x004 | Block attributes (BLKATTR) | R/W | 0x0000_0008 | 20.4.2/20-6 |
| 0x008 | Command argument (CMDARG) | R/W | 0x0000_0000 | 20.4.3/20-7 |
| 0x00C | Command transfer type (XFERTYP) | R/W | 0x0000_0000 | 20.4.4/20-8 |
| 0x010 | Command response0 (CMDRSP0) | R | 0x0000_0000 | 20.4.5/20-11 |
| 0x014 | Command response1 (CMDRSP1) | R | 0x0000_0000 | 20.4.5/20-11 |
| 0x018 | Command response2 (CMDRSP2) | R | 0x0000_0000 | 20.4.5/20-11 |
| 0x01C | Command response3 (CMDRSP3) | R | 0x0000_0000 | 20.4.5/20-11 |
| 0x020 | Data buffer access port (DATPORT) | R/W | 0x0000_0000 | 20.4.6/20-12 |
| 0x024 | Present state (PRSSTAT) | R | 0x*nn*8*n*_00*n*0 | 20.4.7/20-13 |
| 0x028 | Protocol control (PROCTL) | R/W | 0x0000_0000 | 20.4.8/20-17 |
| 0x02C | System control (SYSCTL) | Mixed | 0x0000_8000 | 20.4.9/20-19 |
| 0x030 | Interrupt status (IRQSTAT) | w1c | 0x0000_0000 | 20.4.10/20-22 |
| 0x034 | Interrupt status enable (IRQSTATEN) | R/W | 0x117F_013F | 20.4.11/20-26 |
| 0x038 | Interrupt signal enable (IRQSIGEN) | R/W | 0x0000_0000 | 20.4.12/20-29 |
| 0x03C | Auto CMD12 status (AUTOC12ERR) | R | 0x0000_0000 | 20.4.13/20-31 |
| 0x040 | Host controller capabilities (HOSTCAPBLT) | R | 0x01E3_0000 | 20.4.14/20-33 |
| 0x044[1] | Watermark level (WML) | R/W | 0x0010_0010 | 20.4.15/20-34 |
| 0x050 | Force event (FEVT) | W | 0x0000_0000 | 20.4.16/20-34 |
| 0x0FC | Host controller version (HOSTVER) | R | 0x0000_0001 | 20.4.17/20-36 |
| 0x40C | DMA control register (DCR) | R/W | 0x0000_0000 | 20.4.18/20-37 |

[1] The addresses following 0x044, except 0x050, 0x0FC and 0x40C, are reserved and read as all 0s. Writes to these registers are ignored.

## 20.4.1 DMA System Address Register (DSADDR)

The DMA system address register contains the lower 32-bits of the system memory address used for DMA transfers. Only access this register when no transactions are executing (after transactions have stopped). The host driver should wait until PRSSTAT[DLA] is cleared.

**NOTE**

> This register contains only the lower 32 bits of the DMA address. The 4 high-order bits are in ECMCR[ESDHC_UPRADR]; see Section 23.4.1.26, "ECM Control Register (ECMCR)," for this register.

Offset: 0x000 Access: Read/Write

| | 0–31 |
|---|---|
| R<br>W | DS_ADDR |
| Reset | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 |

**Figure 20-3. DMA System Address Register (DSADDR)**

**Table 20-3. DSADDR Field Descriptions**

| Field | Description |
|---|---|
| 0–31<br>DS_ADDR | DMA system address, lower 32 bits. When the eSDHC stops a DMA transfer, this register points to the system address of the next contiguous data position. The upper four bits of the DMA system address are stored in ECMCR[ESDHC_UPRADR], see Section 23.4.1.26, "ECM Control Register (ECMCR)".<br>**Note:** The DS_ADDR must be aligned to a four-byte boundary; the two least-significant bits must be cleared. |

## 20.4.2 Block Attributes Register (BLKATTR)

The block attributes register configures the number of data blocks and the number of bytes in each block. Only access this register when no transactions are executing (after transactions have stopped). The host driver should wait until PRSSTAT[DLA] is cleared. During a data transfer,

- Reading this register may return an invalid value.
- Writing this register is ignored.

Offset: 0x004 (BLKATTR) Access: Read/Write

| | 0–15 | 16–18 | 19–31 |
|---|---|---|---|
| R<br>W | BLKCNT | — | BLKSZE |
| Reset | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 0 0 | 0 0 0 0 0 0 0 0 0 1 0 0 0 |

**Figure 20-4. Block Attributes Register (BLKATTR)**

**Table 20-4. BLKATTR Field Descriptions**

| Field | Description |
|---|---|
| 0–15<br>BLKCNT | Block count for current transfer. This field is enabled when XFERTYP[BCEN] is set and is valid only for multiple block transfers. The host driver should set this field to a value between 1 and the maximum block count. The eSDHC decrements the block count after each block transfer and stops when the count reaches zero. Clearing this field results in no data blocks being transferred.<br>When saving transfer context as a result of a suspend command, this field indicates the number of blocks yet to be transferred. When restoring transfer context prior to issuing a resume command, the host driver should write the previously saved block count.<br>0000 Stop count<br>0001 1 block<br>0002 2 blocks<br>...<br>FFFF 65,535 blocks |
| 16–18 | Reserved |
| 19–31<br>BLKSIZE | Transfer block size. Specifies the block size for block data transfers. Values can range from one byte up to the maximum buffer size.<br>The DMA always writes at least four bytes to memory. Thus, software should allocate a buffer space rounded up to a 4-byte alighted size in order to avoid data corruption.<br>0000 No data transfer<br>0001 1 byte<br>0002 2 bytes<br>0003 3 bytes<br>0004 4 bytes<br>...<br>01FF 511 bytes<br>0200 512 bytes<br>...<br>0800 2048 bytes<br>1000 4096 bytes |

## 20.4.3 Command Argument Register (CMDARG)

The command argument register contains the SD/MMC command argument.

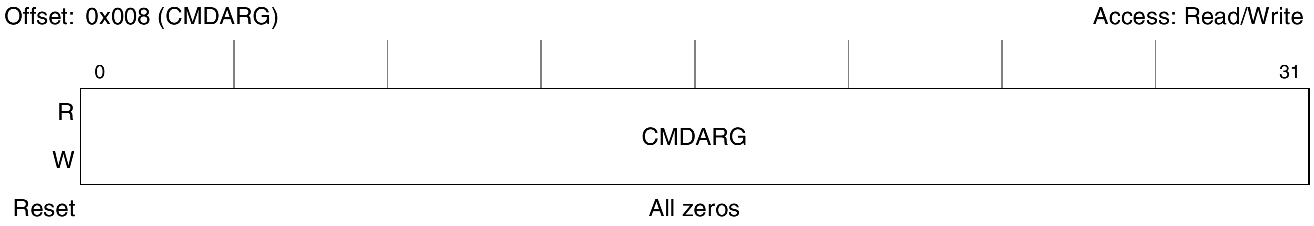


**Figure 20-5. Command Argument Register (CMDARG)**

**Table 20-5. CMDARG Field Descriptions**

| Field | Description |
|---|---|
| 0–31<br>CMDARG | Command argument. The SD/MMC command argument is specified as bits 39–8 of the command format in the SD or MMC Specification. If PRSSTAT[CMD] is set, this register is write-protected. |

### 20.4.4 Transfer Type Register (XFERTYP)

The transfer type register controls the operation of data transfers. The host driver should set this register before issuing a command followed by a data transfer, or before issuing a resume command. To prevent data loss, the eSDHC prevents a write to the bits that are involved in the data transfer of this register while the data transfer is active.

The host driver should check PRSSTAT[CDIHB] and PRSSTAT[CIHB] before writing to this register.

- If PRSSTAT[CDIHB] is set, any attempt to send a command with data by writing to this register is ignored.
- If PRSSTAT[CIHB] is set, any write to this register is ignored.

Offset: 0x00C (XFERTYP) Access: Read/Write

| | 0 1 | 2 7 | 8 9 | 10 | 11 | 12 | 13 | 14 15 | 16 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | CMDINX | CMD TYP | DP SEL | CICEN | CCCEN | — | RSP TYP | — | MSB SEL | DTD SEL | — | AC12 EN | BCEN | DMA EN |
| Reset | All zeros | | | | | | | | | | | | | | |

**Figure 20-6. Transfer Type Register (XFERTYP)**

**Table 20-6. XFERTYP Field Descriptions**

| Field | Description |
|---|---|
| 0–1 | Reserved |
| 2–7<br>CMDINX | Command index. These bits should be set to the command number (CMD0–63, ACMD0–63) that is specified in bits 45–40 of the command format in the *SD Memory Card Physical Layer Specification*. |
| 8–9<br>CMDTYP | Command type. There are three types of special commands: suspend, resume, and abort. Clear this bit field for all other commands.<br>• Suspend command.<br>If the suspend command succeeds, the eSDHC assumes the SD bus has been released and it is possible to issue the next command which uses the SDHC_DAT line. The eSDHC de-asserts read wait for read transactions and stops checking busy for write transactions. In 4-bit mode, the interrupt cycle starts.<br>If the suspend command fails, the eSDHC maintains its current state, and the host driver should restart the transfer by setting PROCTL[CREQ]. The eSDHC does not check if the suspend command succeeds or not. It is the host driver’s responsibility to issue a normal CMD52 marked as suspend command when the suspend request is accepted by the card, so that eSDHC can be informed that the SD bus is released and de-assert read wait during read operation.<br>• Resume command. The host driver restarts the data transfer by restoring the registers saved before sending the suspend command and sends the resume command. The eSDHC checks for pending busy state before starting write transfers.<br>• Abort command.<br>If this command is set when executing a read transfer, the eSDHC stops reads to the buffer.<br>If this command is set when executing a write transfer, the eSDHC stops driving the SDHC_DAT line.<br>After issuing the abort command, the host driver should issue a software reset. (Abort transaction)<br>00 Normal—other commands<br>01 Suspend—CMD52 for writing bus suspend in the common card control register (CCCR)<br>10 Resume—CMD52 for writing function select in CCCR<br>11 Abort—CMD12, CMD52 for writing I/O abort in CCCR |

**Table 20-6. XFERTYP Field Descriptions (continued)**

| Field | Description |
|---|---|
| 10<br>DPSEL | Data present select. Set to indicate that data is present and should be transferred using the SDHC_DAT line. It is cleared for the following:<br>• Commands using only the SDHC_CMD line (e.g. CMD52)<br>• Commands with no data transfer but using busy signal on the SDHC_DAT[0] line (R1b or R5b, e.g. CMD38)<br>**Note:** In resume command, this bit should be set while the other bits in this register should be set the same as when the transfer initially launched.<br>0 No data present<br>1 Data present |
| 11<br>CICEN | Command index check enable.<br>0 Disable. The index field is not checked.<br>1 Enable. The eSDHC checks the index field in the response to see if it has the same value as the command index. If it is not, it is reported as a command index error. |
| 12<br>CCCEN | Command CRC check enable. The number of bits checked by the CRC field value changes according to the length of the response. (Refer to RSPTYP[1:0] and Table 20-8.)<br>0 Disable. The CRC field is not checked.<br>1 Enable. The eSDHC checks the CRC field in the response if it contains the CRC field. If an error is detected, it is reported as a command CRC error. |
| 13 | Reserved |
| 14–15<br>RSPTYP | Response type select.<br>00 No response<br>01 Response length 136<br>10 Response length 48<br>11 Response length 48 check busy after response |
| 16–25 | Reserved |
| 26<br>MSBSEL | Multi/single block select. Enables multiple block SDHC_DAT line data transfers. For any other commands, this bit should be cleared. If this bit is cleared, it is not necessary to set the block count register. (Refer to Table 20-7.)<br>0 Single block<br>1 Multiple blocks |
| 27<br>DTDSEL | Data transfer direction select. Defines the direction of SDHC_DAT line data transfers. The bit is set by the host driver to transfer data from the SD card to the eSDHC and it is cleared for all other commands.<br>0 Write (host to card)<br>1 Read (card to host) |
| 28 | Reserved |
| 29<br>AC12EN | Auto CMD12 enable. Multiple block transfers for memory require CMD12 to stop the transaction. If this bit is set, the eSDHC issues CMD12 automatically when the last block transfer is completed. The host driver should not set this bit to issue commands that do not require CMD12 to stop a multiple block data transfer. In particular, secure commands defined in the Part 3 File Security specification do not require CMD12. In a single block transfer, the eSDHC ignores this bit.<br>0 Disable<br>1 Enable |

Table 20-6. XFERTYP Field Descriptions (continued)

| Field | Description |
|---|---|
| 30<br>BCEN | Block count enable. Enables the block attributes register, which is only relevant for multiple block transfers. When this bit is cleared, the block attributes register is disabled, which is useful in executing an infinite transfer.<br>0 Disable<br>1 Enable |
| 31<br>DMAEN | DMA enable. Enables DMA functionality as described in Section 20.5.2, “DMA CCB Interface.” If this bit is set, a DMA operation should begin when the host driver writes to the CMDINX field of the transfer type register.<br>0 Disable<br>1 Enable |

Table 20-7 shows how register settings determine types of data transfers.

Table 20-7. Determination of Transfer Type

| Multi/Single Block Select XFERTYP[MSBSEL] | Block Count Enable XFERTYP[BCEN] | Block Count BLKATTR[BLKCNT] | Function |
|---|---|---|---|
| 0 | Don’t Care | Don’t Care | Single Transfer |
| 1 | 0 | Don’t Care | Infinite Transfer |
| 1 | 1 | Positive Number | Multiple Transfer |
| 1 | 1 | Zero | No Data Transfer |

Table 20-8 shows how the response type can be determined by the command index check enable, command CRC check enable, and response type bits.

Table 20-8. Relation Between Parameters and Name of Response Type

| Response Type XFERTYP[RSPTYP] | Index Check Enable XFERTYP[CICEN] | CRC Check Enable XFERTYP[CCCEN] | Response Type |
|---|---|---|---|
| 00 | 0 | 0 | No Response |
| 01 | 0 | 1 | R2 |
| 10 | 0 | 0 | R3, R4 |
| 10 | 1 | 1 | R1, R5, R6 |
| 11 | 1 | 1 | R1b, R5b |

## NOTE

The CRC field for R3 and R4 is expected to be all 1s. The CRC check should be disabled for these response types.

## 20.4.5 Command Response 0–3 (CMDRSP0–3)

The command response registers stores the four parts of the response bits from the card.

Offset: 0x010 (CMDRSP0)
0x014 (CMDRSP1)
0x018 (CMDRSP2)
0x01C (CMDRSP3)

Access: Read

| | 0 … 31 |
|---|---|
| R | CMDRSP |
| W | |
| Reset | All zeros |

**Figure 20-7. Command Response 0–3 Register (CMDRSP*n*)**

Table 20-9 describes the mapping of command responses from the SD bus to the command response registers for each response type. In the table, R[ ] refers to a bit range within the response data as transmitted on the SD bus.

**Table 20-9. Response Bit Definition for Each Response Type**

| Response Type | Meaning of Response | Response Field | Response Register |
|---|---|---|---|
| R1,R1b (normal response) | Card status | R[39:8] | CMDRSP0 |
| R1b (Auto CMD12 response) | Card status for Auto CMD12 | R[39:8] | CMDRSP3 |
| R2 (CID, CSD register) | CID/CSD register [127:8] | R[127:8] | {CMDRSP3[23:0], CMDRSP2, CMDRSP1, CMDRSP0} |
| R3 (OCR register) | OCR register for memory | R[39:8] | CMDRSP0 |
| R4 (OCR register) | OCR register for I/O etc. | R[39:8] | CMDRSP0 |
| R6 (publish RCA) | New published RCA[31:16] and card status[15:0] | R[39:8] | CMDRSP0 |

This table shows that:

- Most responses with a length of 48 (R[47:0]) have 32 bits of the response data (R[39:8]) stored in the CMDRSP0 register.
- Responses of type R1b (Auto CMD12 responses) have response data bits R[39:8] stored in the CMDRSP3 register.
- Responses with length 136 (R[135:0]) have 120 bits of the response data (R[127:8]) stored in the CMDRSP0, 1, 2, and 3 registers.

To be able to read the response status efficiently, the eSDHC only stores part of the response data in the command response registers. This enables the host driver to efficiently read 32 bits of response data in one read cycle on a 32-bit bus system. Parts of the response, the index field, and the CRC are checked by the eSDHC (as specified by XFERTYP[CICEN, CCCEN]) and generate an error interrupt if any error is detected. The bit range for the CRC check depends on the response length. If the response length is 48, the eSDHC checks R[47:1], and if the response length is 136, the eSDHC checks R[119:1].

Since the eSDHC may have a multiple block data transfer executing concurrently with a CMD_wo_DAT command, the eSDHC stores the Auto CMD12 response in the CMDRSP3 register and the CMD_wo_DAT response is stored in CMDRSP0. This allows the eSDHC to avoid overwriting the Auto CMD12 response with the CMD_wo_DAT and vice versa. When the eSDHC modifies part of the command response registers it preserves the unmodified bits.

### 20.4.6 Buffer Data Port Register (DATPORT)

The buffer data port register is a 32-bit data port register used to access the internal buffer.

**NOTE**

> When the internal DMA is not enabled and a write transaction is in operation, DATPORT must not be read. DATPORT also must not be used to read (or write) data by the CPU or external DMA if the data will be written (or read) by the eSDHC internal DMA.

Offset: 0x020 (DATPORT) Access: Read/Write

| | 0–31 |
|---|---|
| R<br>W | DATCONT |
| Reset | All zeros |

**Figure 20-8. Buffer Data Port Register (DATPORT)**

**Table 20-10. DATPORT Field Descriptions**

| Field | Description |
|---|---|
| 0–31<br>DATCONT | Data content. The buffer data port register is for 32-bit data access by the CPU or an external DMA. When the internal DMA is enabled, any write to this register is ignored, and a read from this register always yields 0. |

### 20.4.7 Present State Register (PRSSTAT)

PRSSTAT indicates the status of the eSDHC to the host driver.

Offset: 0x024 (PRSSTAT) Access: Read

| | 0 | | | | | | | 7 | 8 | 9 | | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | DLSL | | | | | | | | CLSL | — | | | WPSPL | SDHC_CD | — | CINS |
| W | | | | | | | | | | | | | | | | |
| Reset | *n* | *n* | *n* | *n* | *n* | *n* | *n* | *n* | 1 | 0 | 0 | 0 | *n* | *n* | 0 | *n* |

| | 16 | | | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | — | | | | BREN | BWEN | RTA | WTA | SD<br>OFF | PER<br>OFF | HCK<br>OFF | IPG<br>OFF | — | DLA | CDIHB | CIHB |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | *n* | *n* | *n* | *n* | 0 | 0 | 0 | 0 |

**Figure 20-9. Present State Register (PRSSTAT)**

**Table 20-11. PRSSTAT Field Descriptions**

| Field | Description |
|---|---|
| 0–7<br>DLSL | SDHC_DAT[7:0] line signal level. These bits are used to check the SDHC_DAT line level to recover from errors, and for debugging.This is especially useful in detecting the busy signal level from SDHC_DAT[0]. The reset value is affected by the external pull resistors. By default, read value of this bit field after reset is 11110111, when SDHC_DAT[3] is pull-down and other lines are pull-up. |
| 8<br>CLSL | SDHC_CMD line signal level. This status is used to check the SDHC_CMD line level to recover from errors, and for debugging. The reset value is affected by the external pull resistor, by default, read value of this bit after reset is 1, when the command line is pull-up. |
| 9–11 | Reserved |

| PRSSTAT Bit | SDHC_DAT*n* |
|---|---|
| 0 | 7 |
| 1 | 6 |
| 2 | 5 |
| 3 | 4 |
| 4 | 3 |
| 5 | 2 |
| 6 | 1 |
| 7 | 0 |

**Table 20-11. PRSSTAT Field Descriptions (continued)**

| Field | Description |
|---|---|
| 12<br>WPSPL | Write protect switch pin level. The write protect switch is supported for memory and combo cards.This bit reflects the SDHC_WP pin of the card socket. A software reset does not affect this bit. The reset value is affected by the external write protect switch.<br>If the SDHC_WP pin is not used but is exposed to pins (PMUXCR[SDHC_WP]=1), it should be tied to a value such that write is enabled (0 if GENCFGR[SDHC_WP_INV]=0 or 1 if GENCFGR[SDHC_WP_INV]=1). If the SDHC_WP pin is not exposed to pins (PMUXCR[SDHC_WP]=0), then the value of the SDHC_WP pin does not affect this register field. However, if eSDHC write functionality is required, then in this case GENCFGR[SDHC_WP_INV] should be set to 1.<br>See Section 23.4.1.8, "General Configuration Register (GENCFGR)," for information on the SDHC_WP_INV bit.<br>0 Write protected (SDHC_WP = 1 if GENCFGR[SDHC_WP_INV]=0 or SDHC_WP = 0 if GENCFGR[SDHC_WP_INV]=1)<br>1 Write enabled (SDHC_WP = 0 if GENCFGR[SDHC_WP_INV]=0 or SDHC_WP = 1 if GENCFGR[SDHC_WP_INV]=1 or PMUXCR[SDHC_WP]=0 and GENCFGR[SDHC_WP_INV]=1). |
| 13<br>SDHC_CD | Card detect pin level. This bit reflects the inverse value of the $\overline{\text{SDHC\_CD}}$ pin for the card socket. Debouncing is not performed on this bit. This bit may be valid, but it is not guaranteed because of a propagation delay. Use of this bit is limited to testing since it must be debounced by software. A software reset does not affect this bit. Write to the force event register does not affect this bit. If PMUXCR[SDHC_CD]=1, the reset value of this field is affected by the external card detection pin; if this bit is not used, it should be tied to 0. If PMUXCR[SDHC_CD]=0, this field is unaffected by the external card detect pin, and will permanently indicate that a card is present.<br>0 No card present ($\overline{\text{SDHC\_CD}}$ = 1) and PMUXCR[SDHC_CD]=1)<br>1 Card present ($\overline{\text{SDHC\_CD}}$ = 0 or PMUXCR[SDHC_CD]=0) |
| 14 | Reserved |
| 15<br>CINS | Card inserted. Indicates if a card has been inserted. The eSDHC debounces this signal so that the host driver does not need to wait for it to stabilize. Changing from 0 to 1 generates a card-insertion interrupt in the interrupt status register and changing from 1 to 0 generates a card removal interrupt in the interrupt status register. A write to the force event register does not affect this bit.<br>The software reset for all in the system control register does not affect this bit. A software reset does not affect this bit.<br>0 Power-on-reset or no card<br>1 Card inserted |
| 16–19 | Reserved |
| 20<br>BREN | Buffer read enable. This status is used for non-DMA read transfers. The eSDHC may implement multiple buffers to transfer data efficiently. This read-only flag indicates that a burst-length of valid data exists in the host-side buffer. When the buffer is read, this bit is cleared. When a burst length of data is ready in the buffer, this bit is set and a buffer read ready interrupt is generated (if the interrupt is enabled).<br>0 Buffer read disable<br>1 Buffer read enable |
| 21<br>BWEN | Buffer write enable. This status is used for non-DMA write transfers. The eSDHC can implement multiple buffers to transfer data efficiently. This read-only flag indicates if space is available for a burst length of write data.<br>When the buffer is written, this bit is cleared. When a burst length of data is written to the buffer, this bit is set and a buffer write ready interrupt is generated (if the interrupt is enabled).<br>0 Buffer write disable<br>1 Buffer write enable |

**Table 20-11. PRSSTAT Field Descriptions (continued)**

| Field | Description |
|---|---|
| 22<br>RTA | Read transfer active. This status is used for detecting completion of a read transfer.<br>This bit is set for either of the following conditions:<br>• After the end bit of the read command<br>• When writing a 1 to PROCTL[CREQ] to restart a read transfer<br>This bit is cleared for either of the following conditions:<br>• When the last data block as specified by block length is transferred to the system<br>• When all valid data blocks have been transferred to the system and no current block transfers are being sent as a result of PROCTL[SABGREQ] being set. A transfer complete interrupt is generated when this bit changes to 0.<br>0 No valid data<br>1 Transferring data |
| 23<br>WTA | Write transfer active. This status indicates a write transfer is active. If this bit is 0, it means no valid write data exists in eSDHC.<br>This bit is set in either of the following cases:<br>• After the end bit of the write command.<br>• When writing a 1 to PROCTL[CREQ] to restart a write transfer.<br>This bit is cleared in either of the following cases:<br>• After getting the CRC status of the last data block, as specified by the transfer count (single and multiple)<br>• After getting the CRC status of any block where data transmission is about to be stopped by a stop-at-block-gap request.<br>During a write transaction, a IRQSTAT[BGE] interrupt is generated when this bit is changed to 0, as result of PROCTL[SABGREQ] being set. This status is useful for the host driver in determining when to issue commands during write busy.<br>0 No valid data<br>1 Transferring data |
| 24<br>SDOFF | SD clock gated off internally. Indicates the SD clock is internally gated off because of a buffer overrun, buffer underrun, or a read pause without read-wait assertion. This bit is for the host driver to debug data transaction on SD bus.<br>This status bit resets to 0, but reflects the value of the automatic clock gating and may transition to 1 if the eSDHC is idle. |
| 25<br>PEROFF | The internal bus clock gated off internally. This status bit indicates the internal bus clock is internally gated off. This bit is for the host driver to debug a transaction on SD bus.<br>This status bit resets to 0, but reflects the value of the automatic clock gating and may transition to 1 if the eSDHC is idle. |
| 26<br>HCKOFF | Master clock gated off internally. This status bit indicates master clock is internally gated off. This bit is for the host driver to debug a data transfer.<br>This status bit resets to 0, but reflects the value of the automatic clock gating and may transition to 1 if the eSDHC is idle. |
| 27<br>IPGOFF | Controller clock gated off internally. Indicates that the controller clock is internally gated off. This bit is for the host driver to debug.<br>This status bit resets to 0, but reflects the value of the automatic clock gating and may transition to 1 if the eSDHC is idle. |
| 28 | Reserved |

**Table 20-11. PRSSTAT Field Descriptions (continued)**

| Field | Description |
|---|---|
| 29<br>DLA | Data line active. Indicates whether one of the SDHC_DAT line on SD bus is in use.<br><br>For read transactions, this bit indicates if a read transfer is executing on the SD bus. Clearing this bit from 1 to 0 between data blocks generates a block gap event interrupt.<br>This bit is set in either of the following cases:<br>• After the end bit of the read command<br>• When writing a 1 to PROCTL[CREQ] to restart a read transfer<br>This bit is cleared in either of the following cases:<br>• When the end bit of the last data block is sent from the SD bus to the eSDHC<br>• When beginning a read wait transfer initiated by a stop at block gap request<br><br>The eSDHC waits at the next block gap by driving read wait at the start of the interrupt cycle. If the read-wait signal is already driven (data buffer cannot receive data), the eSDHC can wait for current block gap by continuing to drive the read-wait signal. It is necessary to support read wait in order to use the suspend/resume function.<br><br>For write transactions, this bit indicates that a write transfer is executing on the SD bus. Clearing this bit from 1 to 0 generates a transfer complete interrupt.<br>This bit is set in any of the following cases:<br>• After the end bit of the write command<br>• When writing a 1 to PROCTL[CREQ] to continue a write transfer<br>This bit is cleared in any of the following cases:<br>• When the SD card releases write-busy of the last data block, the eSDHC also detects if output is not busy. If the SD card does not drive the busy signal after CRC status is received, the eSDHC should consider the card drive not busy.<br>• When the SD card releases write-busy prior to waiting for write transfer as a result of a stop at block gap request<br><br>0 SDHC_DAT line inactive<br>1 SDHC_DAT line active |
| 30<br>CDIHB | Command inhibit (SDHC_DAT). This bit is set if the SDHC_DAT line is active, the read transfer active is set, or read wait is asserted. If this bit is cleared, it indicates the eSDHC can issue the next SD/MMC command. Commands with busy signal belong to command inhibit (SDHC_DAT) (e.g. R1b and R5b type). Clearing from 1 to 0 generates a transfer complete interrupt.<br>**Note:** The SD host driver can save registers for a suspend transaction after this bit has cleared from 1 to 0.<br>0 Can issue command which uses the SDHC_DAT line<br>1 Cannot issue command which uses the SDHC_DAT line |
| 31<br>CIHB | Command inhibit (SDHC_CMD). This bit is cleared, if the SDHC_CMD line is not in use and the eSDHC can issue a SD/MMC command using the SDHC_CMD line.<br>This bit is set immediately after the XFERTYP register is written. This bit is cleared when the command response is received. Even if the CDIHB bit is set, commands using only the SDHC_CMD line can be issued if this bit is cleared. Clearing from 1 to 0 generates a command complete interrupt.<br>If the eSDHC cannot issue the command because of a command conflict error (refer to command CRC error) or because of command not issued by Auto CMD12 error, this bit remains set and IRQSTAT[CC] is not set. Status issuing Auto CMD12 is not read from this bit.<br>0 Can issue command using only SDHC_CMD line<br>1 Cannot issue command |

### NOTE

The host driver can issue CMD0, CMD12, CMD13 (for memory) when the SDHC_DAT lines are busy during a data transfer. These commands can be issued when PRSSTAT[CIHB] is cleared. Other commands should be issued when PRSSTAT[CDIHB] is cleared. Possible changes to the SD physical specification may add other commands to this list in the future.

## 20.4.8 Protocol Control Register (PROCTL)

The protocol control register is shown in Figure 20-10.

Offset: 0x028 (PROCTL) Access: Read/Write

| | 0 | | | | 4 | 5 | 6 | 7 | 8 | | | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | | | | | WE CRM | WE CINS | WE CINT | — | | | | | RW CTL | CREQ | SABG REQ |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 16 | | | | | | | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | | | | | | | | CDSS | CDTL | EMODE | | D3CD | DTW | | — |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

**Figure 20-10. Protocol Control Register (PROCTL)**

**Table 20-12. PROCTL Field Descriptions**

| Field | Description |
|---|---|
| 0–4 | Reserved |
| 5<br>WECRM | Wake-up event enable on SD card removal. This bit enables wakeup event via card removal assertion in the IRQSTAT register. FN_WUS (wake-up support) in CIS does not affect this bit.<br>0 Disable<br>1 Enable |
| 6<br>WECINS | Wake-up event enable on SD card insertion. This bit enables wakeup event via card insertion assertion in the IRQSTAT register. FN_WUS (wake-up support) in CIS does not affect this bit.<br>0 Disable<br>1 Enable |
| 7<br>WECINT | Wake-up event enable on card interrupt. This bit enables wakeup event via card interrupt assertion in the IRQSTAT register. This bit can be set to 1 if FN_WUS (wake-up support) in CIS is set to 1.<br>0 Disable<br>1 Enable |
| 8–12 | Reserved |

### Table 20-12. PROCTL Field Descriptions (continued)

| Field | Description |
|---|---|
| 13<br>RWCTL | Read wait control.<br>If the card supports read wait, set this bit to enable the read wait protocol to stop read data using the SDHC_DAT[2] line. Otherwise, the eSDHC has to stop the SD clock to hold read data, which restricts command generation.<br>If the card does not support read wait, this bit should never be set otherwise an SDHC_DAT line conflict may occur. If this bit is cleared, a stop-at-block-gap-during-read operation is also supported, but the eSDHC stops the SD clock to pause the reading operation.<br>0 Disable read-wait control, and stop SD clock at block gap when the SABGREQ bit is set<br>1 Enable read-wait control, and assert read wait without stopping the SD clock at block gap when PROCTL[SABGREQ] is set |
| 14<br>CREQ | Continue request. Restarts a transaction which was stopped using the stop-at-block-gap request. To cancel the request, clear SABGREQ and set this bit to restart the transfer.<br>The eSDHC automatically clears this bit in either of the following cases:<br>• For a read transaction, the PRSSTAT[DLA] bit changes from 0 to 1 as a read transaction restarts.<br>• For a write transaction, the PRSSTAT[WTA] bit changes from 0 to 1 as the write transaction restarts.<br>Therefore, it is not necessary for the host driver to clear. If SABGREQ and this bit are set, the continue request is ignored.<br>0 No effect<br>1 Restart |
| 15<br>SABGREQ | Stop at block gap request. Stops executing a transaction at the next block gap for both DMA and non-DMA transfers. Until the TC bit is set, indicating a transfer completion, the host driver should leave this bit set. Clearing SABGREQ and CREQ does not cause the transaction to restart.<br>Read wait is used to stop the read transaction at the block gap. The eSDHC honors stop-at-block-gap request for write transfers. Otherwise, the eSDHC stops the SD bus clock to pause the read operation during the block gap.<br><br>For write transfers in which the host driver writes data to the data port register, the host driver should set this bit after all block data is written. If this bit is set, the host driver should not write data to the DATPORT register after a block is sent. When this bit is set, the host driver should not clear this bit before IRQSTAT[TC] is set. Otherwise, the eSDHC behavior is undefined. Confirm that IRQSTAT[TC] is enabled.<br>This bit affects PRSSTAT[RTA, WTA, DLA, CIHB].<br>0 Transfer<br>1 Stop or not resume yet |
| 16–23 | Reserved |
| 24<br>CDSS | Card detect signal selection. Selects the source for card detection.<br>0 Card detection level is selected (for normal purpose)<br>1 Card detection test level is selected (for test purpose) |
| 25<br>CDTL | Card detect test level. Determines card insertion status when CDSS is set.<br>0 No card in the slot<br>1 Card is inserted |
| 26–27<br>EMODE | Endian mode. eSDHC supports only address-invariant mode in data transfer.<br>00 Reserved<br>01 Reserved<br>10 Address-invariant mode. Each byte location in the main memory is mapped to the same byte location in the MMC/SD card.<br>11 Reserved |

**Table 20-12. PROCTL Field Descriptions (continued)**

| Field | Description |
|---|---|
| 28<br>D3CD | SDHC_DAT3 as card detection pin. If this bit is set, SDHC_DAT3 should be pulled down to act as a card detection pin. Be cautious when using this feature, because SDHC_DAT3 is chip-select for SPI mode, and a pull-down on this pin and CMD0 may set the card into SPI mode, which the eSDHC does not support.<br>0 SDHC_DAT3 does not monitor card insertion<br>1 SDHC_DAT3 is card detection pin |
| 29–30<br>DTW | Data transfer width. Selects the data width of the SD bus. The host driver should set it to match the data width of the card.<br>00 1-bit mode<br>01 4-bit mode<br>10 8-bit mode<br>11 Reserved |
| 31 | Reserved |

There are three ways to restart the transfer after a stop at the block gap. The appropriate method depends on whether the eSDHC issues a suspend command or the SD card accepts the suspend command:

- If the host driver does not issue a suspend command, the continue request should be used to restart the transfer.
- If the host driver issues a suspend command and the SD card accepts it, a resume command should be used to restart the transfer.
- If the host driver issues a suspend command and the SD card does not accept it, PROCTL[CREQ] should be used to restart the transfer.

Any time PROCTL[SABGREQ] stops the data transfer, the host driver should wait for IRQSTAT[TC] before attempting to restart the transfer. When restarting the data transfer by continue request, the host driver should clear PROCTL[SABGREQ] before or simultaneously.

## 20.4.9 System Control Register (SYSCTL)

The system control register is shown in Figure 20-11.

Offset: 0x02C (SYSCTL) Access: Mixed

| | 0 | | | 3 | 4 | 5 | 6 | 7 | 8 | | | 11 | 12 | | | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | — | | | | INITA | | | | — | | | | DTOCV | | | |
| W | | | | | | RSTD | RSTC | RSTA | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 16 | | | | | | | 23 | 24 | | | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | SDCLKFS | | | | | | | | DVS | | | | — | PEREN | HCKEN | IPGEN |
| W | | | | | | | | | | | | | | | | |
| Reset | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Figure 20-11. System Control Register (SYSCTL)**

### Table 20-13. SYSCTL Field Descriptions

| Field | Description |
|---|---|
| 0–3 | Reserved |
| 4<br>INITA | Initialization active. When this bit is written ‘1’, 80 SD clocks are sent to the card. After the 80 clocks are sent, this bit is self-cleared. This bit is very useful during the card power-up period when 74 SD clocks are needed and clock auto-gating feature is enabled.<br>Writing one to this bit when it is already set has no effect. Clearing this bit at any time does not affect it. When PRSSTAT[CIHB] or PRSSTAT[CDIHB] is set, writing a one to this bit is ignored. That is, when the command line or data line is active, writing to this bit is not allowed. |
| 5<br>RSTD | Software reset for SDHC_DAT line. The DMA and part of the data circuit are reset. The following registers and bits are cleared by this bit:<br>• DATPORT register<br>• Buffer is cleared and initialized; PRSSTAT register<br>• PRSSTAT[BREN, BWEN, RTA, WTA, DLA, CDIHB]<br>• PROCTL[CREQ, SABGREQ]<br>• IRQSTAT[BRR, BWR, DINT, BGE, TC]<br>• DSADDR<br>• BLKATTR<br>• PROCTL[IABG, RWCTL, DTW]<br>• IRQSTAT[DMAE, DEBE, DCE, DTOE]<br>• IRQSTATEN[DMASEN, DEBESEN, DCESEN, DTOESEN, BRRSEN, BWRSEN, DINTSEN, BGESEN, TCSEN]<br>• IRQSIGEN[DMAEIEN, DEBEIEN, DCEIEN, DTOEIEN, BRRIEN, BWRIEN, DINTIEN, BGEIEN, TCIEN]<br>• WML<br>0 Work<br>1 Reset |
| 6<br>RSTC | Software reset for SDHC_CMD line. Only part of the command circuit is reset. The following registers and bits are cleared by this bit:<br>• PRSSTAT[CIHB]<br>• IRQSTAT[CC]<br>• CMDARG<br>• XFERTYP<br>• CMDRSP0<br>• CMDRSP1<br>• CMDRSP2<br>• CMDRSP3<br>• PRSSTAT[CDIHB, CIHB]<br>• IRQSTAT[AC12E, CIE, CEBE, CCE, CTOE]<br>• IRQSTATEN[AC12ESEN, CIESEN, CEBESEN, CCESEN, CTOESEN, CCSEN]<br>• IRQSIGEN[AC12EIEN, CIEIEN, CEBEIEN, CCEIEN, CTOEIEN, CCIEN]<br>• AUTOC12ERR<br>0 Work<br>1 Reset |
| 7<br>RSTA | Software reset for all. This reset affects the entire host controller except for the card-detection circuit. Register bits of type ROC, RW, RW1C, and RWAC are cleared.<br>During its initialization, the host driver should set this bit to reset the eSDHC. The eSDHC should clear this bit when capabilities registers are valid and the host driver can read them. Additional use of the this bit does not affect the value of the capabilities registers. After this bit is set, it is recommended the host driver reset the external card and re-initialize it.<br>0 Work<br>1 Reset |

**Table 20-13. SYSCTL Field Descriptions (continued)**

| Field | Description |
|---|---|
| 8–11 | Reserved |
| 12–15<br>DTOCV | Data timeout counter value. Determines the interval by which SDHC_DAT line timeouts are detected. Refer to the data timeout error Section 20.4.10, “Interrupt Status Register (IRQSTAT)”, for information on factors that dictate timeout generation. Timeout clock frequency is generated by dividing the base clock SDHC_CLK value by this value. When setting this register, prevent inadvertent timeout events by clearing IRQSTATEN[DTOESEN].<br>0000 SDHC_CLK x $2^{13}$<br>0001 SDHC_CLK x $2^{14}$<br>...<br>1110 SDHC_CLK x $2^{27}$<br>1111 Reserved |
| 16–23<br>SDCLKFS | SDHC_CLK frequency select. This field, together with DVS, selects the frequency of SDHC_CLK pin. This bit holds the prescaler of the base clock frequency. Only the following settings are allowed:<br>0x01 Base clock divided by 2<br>0x02 Base clock divided by 4<br>0x04 Base clock divided by 8<br>0x08 Base clock divided by 16<br>0x10 Base clock divided by 32<br>0x20 Base clock divided by 64<br>0x40 Base clock divided by 128<br>0x80 Base clock divided by 256<br>Multiple bits must not be set or the behavior of this prescaler is undefined.<br>According to the SD Physical Specification version 1.1, the maximum SD clock frequency is 50 MHz, and should never exceed this limit. The frequency of SDHC_CLK is set by the following formula:<br>$\text{clock frequency} = (\text{base clock}) / [(\text{SDCLKFS} \times 2) \times (\text{DVS} + 1)]$ *Eqn. 20-1*<br>For example, if the base clock frequency is 96 MHz, and the target frequency is 25 MHz, then choosing the prescaler value of 0x1 and divisor value of 0x1 yields 24 MHz, which is the nearest frequency less than or equal to the target. Similarly, to approach a clock value of 400 KHz, the prescaler value of 0x04 and divisor value of 0xE yields the exact clock value of 400 KHz.<br>The reset value of this bit field is 0x80. So, if the input base clock is about 96 MHz, the default SD clock after reset is 375 KHz.<br>**Note:** The base clock frequency equals the platform clock/2. |
| 24–27<br>DVS | Divisor. Provides a more exact divisor to generate the desired SD clock frequency. The settings are as follows:<br>0x0 Divide by 1<br>0x1 Divide by 2<br>...<br>0xE Divide by 15<br>0xF Divide by 16 |
| 28 | Reserved |

Table 20-13. SYSCTL Field Descriptions (continued)

| Field | Description |
|---|---|
| 29<br>PEREN | Peripheral clock enable. If set, the peripheral clock is always active and no automatic gating is applied, thus SDHC_CLK is active only except auto gating-off during buffer danger. If cleared, the peripheral clock is automatically off when no transaction is on the SD bus. Clearing this bit does not stop SDHC_CLK immediately. The peripheral clock will be internally gated off, if none of the following factors are met:<br>• Command part is reset<br>• Data part is reset<br>• Soft reset<br>• Command is about to send<br>• Clock divisor is just updated<br>• Continue request is just set<br>• This bit is set<br>• Card insertion is detected<br>• Card removal is detected<br>• Card external interrupt is detected<br>• 80 clocks for initialization phase is ongoing<br>0 The peripheral clock is internally gated off<br>1 The peripheral clock is not automatically gated off |
| 30<br>HCKEN | Master clock enable. If set, master clock is always active and no automatic gating is applied. If cleared, master clock is automatically off when no data transfer is on SD bus.<br>**Note:** Master clock is the clock to the DMA engine and to the system bus interface logic.<br>0) Master clock is internally gated off<br>1) Master clock is not automatically gated off |
| 31<br>IPGEN | Controller clock enable. If this bit is set, the controller clock is always active and no automatic gating is applied. The controller clock is internally gated off, if neither the following factors is met:<br>• Command part is reset<br>• Data part is reset<br>• Soft reset<br>• Command is about to send<br>• Clock divisor is just updated<br>• Continue request is just set<br>• This bit is set<br>• Card insertion is detected<br>• Card removal is detected<br>• Card external interrupt is detected<br>• The controller clock is not gated off<br>**Note:** The controller clock is not auto-gated off if the peripheral clock is not gated off. So, clearing this bit only does not take effect if SYSCTL[PEREN] is not cleared.<br>0 The controller clock is internally gated off<br>1 The controller clock is not automatically gated off |

## 20.4.10 Interrupt Status Register (IRQSTAT)

An interrupt is generated when one of the status bits and its corresponding interrupt enable bit are set. For all bits, writing one to a bit clears it, while writing zero keeps the bit unchanged. More than one status can be cleared with a single register write. For a card interrupt (IRQSTAT[CINT]), the card must stop asserting the interrupt before writing one to clear. Otherwise, the CINT bit is set again.

Offset: 0x030 (IRQSTAT) Access: w1c

| | 0 2 | 3 | 4 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | — | DMAE | — | AC12E | — | DEBE | DCE | DTOE | CIE | CEBE | CCE | CTOE |
| W | | w1c | | w1c | | w1c | w1c | w1c | w1c | w1c | w1c | w1c |
| Reset | All zeros | | | | | | | | | | | |

| | 16 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|
| R | — | CINT | CRM | CINS | BRR | BWR | DINT | BGE | TC | CC |
| W | | w1c | w1c | w1c | w1c | w1c | w1c | w1c | w1c | w1c |
| Reset | All zeros | | | | | | | | | |

**Figure 20-12. Interrupt Status Register (IRQSTAT)**

**Table 20-14. IRQSTAT Field Descriptions**

| Field | Description |
|---|---|
| 0–2 | Reserved |
| 3<br>DMAE | DMA error. Occurs when internal DMA transfer failed. This bit is set when some error occurs in the data transfer. The value in the DMA system address register is the next fetch address where the error occurs. Since any error corrupts the entire data block, the host driver should restart the transfer from the corrupted block boundary. The address of the block boundary can be calculated from the current DS_ADDR value or the remaining number of blocks and the block size.<br>0 No Error<br>1 Error |
| 4–6 | Reserved |
| 7<br>AC12E | Auto CMD12 error. Occurs when one of the bits in AUTOC12ERR is set. This bit is also set when Auto CMD12 is not executed due to a previous command error.<br>0 No Error<br>1 Error |
| 8 | Reserved |
| 9<br>DEBE | Data end bit error. Occurs when detecting 0 at the end bit position of read data on the SDHC_DAT line or at the end bit position of the CRC.<br>0 No Error<br>1 Error<br>**Note:** When DEBE and CINT are set, the software should ignore DEBE. But, it must not ignore the other status bits. The software should also clear this bit by writing 1 to it. It is highly recommended to clear this bit before the next transfer. |
| 10<br>DCE | Data CRC error. Occurs when detecting CRC error when transferring read data on the SDHC_DAT line or when detecting the write CRC status having a value other than 0b010.<br>0 No Error<br>1 Error |

**Table 20-14. IRQSTAT Field Descriptions (continued)**

| Field | Description |
|---|---|
| 11<br>DTOE | Data timeout error. Occurs during one of following timeout conditions:<br>• Busy timeout for R1b and R5b types<br>• Busy timeout after write CRC status<br>• Read data timeout<br>0 No error<br>1 Timeout |
| 12<br>CIE | Command index error. Occurs if a command index error occurs in the command response.<br>0 No error<br>1 Timeout |
| 13<br>CEBE | Command end bit error. Occurs when the end bit of a command response is 0.<br>0 No error<br>1 End bit error generated |
| 14<br>CCE | Command CRC error. A command CRC error is generated in two cases:<br>• If a response is returned and IRQSTAT[CTOE] is cleared (indicating no timeout), this bit is set when detecting a CRC error in the command response.<br>• The eSDHC detects a SDHC_CMD line conflict by monitoring the SDHC_CMD line when a command is issued. If the eSDHC drives the SDHC_CMD line to 1, but detects 0 on the SDHC_CMD line at the next SDHC_CLK edge, then the eSDHC aborts the command (stop driving SDHC_CMD line) and sets this bit. The CTOE bit is also set to distinguish the SDHC_CMD line conflict.<br>0 No error<br>1 CRC error generated |
| 15<br>CTOE | Command timeout error. Occurs if no response is returned within 64 SDHC_CLK cycles from the end bit of the command. Also, if eSDHC detects a SDHC_CMD line conflict, this bit is set along with IRQSTAT[CCE] as shown in Table 20-27.<br>0 No error<br>1 Time out |
| 16–22 | Reserved |
| 23<br>CINT | Card interrupt.<br>• In 1-bit mode, the eSDHC detects the card interrupt without the SD clock to support wakeup.<br>• In 4-bit mode, the card interrupt signal is sampled during the interrupt cycle. So, there are some sample delays between the interrupt signal from the SD card and the interrupt to the host system.<br>Writing 1 clears this bit. But, if the interrupt source from the SD card is not cleared, this bit is set again. To clear this bit, the SD card interrupt source must be cleared followed by writing 1 to this bit.<br><br>When this bit is set and the host driver needs to start the interrupt service, IRQSIGEN[CINTIEN] should be cleared to stop driving the interrupt signal to the host system. After completing the card interrupt service, write 1 to clear this bit, set IRQSIGEN[CINTIEN], and start sampling the interrupt signal again.<br><br>0 No card interrupt<br>1 Generate card interrupt |
| 24<br>CRM | Card removal. This bit is set if PRSSTAT[CINS] changes from 1 to 0. When the host driver writes 1 to this bit to clear it, the status of PRSSTAT[CINS] should be confirmed. Because the card-detect state may be changed when the host driver clears this bit, an interrupt event may not be generated.<br>When this bit is cleared, it is set again if no card is inserted. To leave it cleared, clear IRQSTATEN[CRMSEN].<br>0 Card state unstable or inserted<br>1 Card removed |

**Table 20-14. IRQSTAT Field Descriptions (continued)**

| Field | Description |
|---|---|
| 25<br>CINS | Card insertion. This bit is set if PRSSTAT[CINS] changes from 0 to 1. When the host driver writes 1 to this bit to clear it, the status of PRSSTAT[CINS] should be confirmed. Because the card-detect state may be changed when the host driver clears this bit, an interrupt event may not be generated.<br>When this bit is cleared, it is set again if a card has been inserted. To leave it cleared, clear IRQSTATEN[CINSEN].<br>0 Card state unstable or removed<br>1 Card inserted |
| 26<br>BRR | Buffer read ready. This bit is set if PRSSTAT[BREN] changes from 0 to 1.<br>0 Not ready to read buffer<br>1 Ready to read buffer |
| 27<br>BWR | Buffer write ready. This bit is set if PRSSTAT[BWEN] changes from 0 to 1.<br>0 Not ready to write buffer<br>1 Ready to write buffer |
| 28<br>DINT | DMA interrupt. Occurs when the internal DMA finishes the data transfer successfully. If errors occur during data transfer, this bit is not set. Instead, the DMAE bit is set.<br>0 No DMA interrupt<br>1 DMA interrupt is generated |
| 29<br>BGE | Block gap event. If PROCTL[SABGREQ] is set, this bit is set when a read or write transaction is stopped at a block gap. If PROCTL[SABGREQ] is cleared, this bit is not set.<br>During a read transaction, this bit is set at the falling edge of the SDHC_DAT line active status (when the transaction is stopped at SD bus timing). Read wait must be supported to use this function.<br>During a write transaction, this bit is set at the falling edge of PRSSTAT[WTA] (after reading the CRC status at SD bus timing).<br>0 No block gap event<br>1 Transaction stopped at block gap |
| 30<br>TC | Transfer complete. This bit is set when a read or write transfer is completed.<br>For a read transaction, this bit is set at the falling edge of PRSSTAT[WTA]. There are two cases in which this interrupt is generated:<br>• When a data transfer is completed, as specified by data length (after the last data has been read to the host system).<br>• When data has stopped at the block gap and completed the data transfer by setting PROCTL[SABGREQ] (after valid data has been read to the host system).<br>For a write transaction, this bit is set at the falling edge of PRSSTAT[DLA]. There are two cases in which this interrupt is generated:<br>• When the last data is written to the SD card, as specified by data length and the busy signal is released.<br>• When data transfers are stopped at the block gap by setting PROCTL[SABGREQ] and data transfers have completed (after valid data is written to the SD card and the busy signal is released). |
| 31<br>CC | Command complete. This bit is set when the end bit of the command response is received (except Auto CMD12). Refer to PRSSTAT[CIHB].<br>0 No command complete<br>1 Command complete |

Table 20-15 below shows that command timeout error has higher priority than command complete. If both bits are set, it can be assumed that the response was not received correctly.

Table 20-15. Relation Between Command Timeout Error and Command Complete Status

| Command Complete | Command Timeout Error | Meaning of the Status |
|---|---|---|
| 0 | 0 | — |
| Don’t Care | 1 | Response not received within 64 SDHC_CLK cycles |
| 1 | 0 | Response received |

Table 20-16 below shows that transfer complete has higher priority than data timeout error. If both bits are set, the data transfer can be considered complete.

Table 20-16. Relation Between Data Timeout Error and Transfer Complete Status

| Transfer Complete | Data Timeout Error | Meaning of the Status |
|---|---|---|
| 0 | 0 | — |
| 0 | 1 | Timeout occur during transfer |
| 1 | X | Data transfer complete |

The relation between command CRC error and command timeout error is shown in Table 20-17 below.

Table 20-17. Relation Between Command CRC Error and Command Timeout Error

| Command CRC Error | Command Timeout Error | Meaning of the Status |
|---|---|---|
| 0 | 0 | No error |
| 0 | 1 | Response Timeout Error |
| 1 | 0 | Response CRC Error |
| 1 | 1 | SDHC_CMD line conflict |

## 20.4.11 Interrupt Status Enable Register (IRQSTATEN)

Setting the bits of IRQSTATEN enables the corresponding interrupt status bit to be set by the specified event. If any bit is cleared, the corresponding IRQSTAT bit is also cleared and is never set.

Offset: 0x034 (IRQSTATEN)

Access: Read/Write

| | 0 | | 2 | 3 | 4 | | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | | | DMAE SEN | — | | | AC12E SEN | — | DEBE SEN | DCE SEN | DTOE SEN | CIE SEN | CEBE SEN | CCE SEN | CTOE SEN |
| Reset | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

| | 16 | | | | | | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | | | | | | | CINT SEN | CRM SEN | CINS SEN | BRR SEN | BWR SEN | DINT SEN | BGE SEN | TC SEN | CC SEN |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |

**Figure 20-13. Interrupt Status Enable Register (IRQSTATEN)**

**Table 20-18. IRQSTATEN Field Descriptions**

| Field | Description |
|---|---|
| 0–2 | Reserved |
| 3<br>DMAESEN | DMA error status enable<br>0 Masked<br>1 Enabled |
| 4–6 | Reserved |
| 7<br>AC12ESEN | Auto CMD12 error status enable<br>0 Masked<br>1 Enabled |
| 8 | Reserved |
| 9<br>DEBESEN | Data end bit error status enable<br>0 Masked<br>1 Enabled |
| 10<br>DCESEN | Data CRC error status enable<br>0 Masked<br>1 Enabled |
| 11<br>DTOESEN | Data timeout error status enable<br>0 Masked<br>1 Enabled |
| 12<br>CIESEN | Command index error status enable<br>0 Masked<br>1 Enabled |
| 13<br>CEBESEN | Command end bit error status enable<br>0 Masked<br>1 Enabled |
| 14<br>CCESEN | Command CRC error status enable<br>0 Masked<br>1 Enabled |

Table 20-18. IRQSTATEN Field Descriptions (continued)

| Field | Description |
|---|---|
| 15<br>CTOESEN | Command timeout error status enable<br>0 Masked<br>1 Enabled |
| 16–22 | Reserved |
| 23<br>CINTSEN | Card interrupt status enable. If this bit is cleared, the eSDHC clears the interrupt request to the system. The card interrupt detection is stopped when this bit is cleared and restarted when this bit is set. To prevent inadvertent interrupts, the host driver should clear this bit before servicing the card interrupt and should set this bit again after all interrupt requests from the card are cleared.<br>0 Masked<br>1 Enabled |
| 24<br>CRMSEN | Card removal status enable<br>0 Masked<br>1 Enabled |
| 25<br>CINSEN | Card insertion status enable<br>0 Masked<br>1 Enabled |
| 26<br>BRRSEN | Buffer read ready status enable<br>0 Masked<br>1 Enabled |
| 27<br>BWRSEN | Buffer write ready status enable<br>0 Masked<br>1 Enabled |
| 28<br>DINTSEN | DMA interrupt status enable<br>0 Masked<br>1 Enabled |
| 29<br>BGESEN | Block gap event status enable<br>0 Masked<br>1 Enabled |
| 30<br>TCSEN | Transfer complete status enable<br>0 Masked<br>1 Enabled |
| 31<br>CCSEN | Command complete status enable<br>0 Masked<br>1 Enabled |

## NOTE

The eSDHC may sample the card interrupt signal during the interrupt period and hold its value in the flip-flop. As a result of synchronization, there is a delay in the card interrupt (which is asserted from the card) to the time the host system is informed.

To detect a SDHC_CMD line conflict, the host driver must set both CTOESEN and CCESEN bits.

## 20.4.12 Interrupt Signal Enable Register (IRQSIGEN)

IRQSIGEN selects which interrupt status is indicated to the host system as the interrupt. These status bits all share the same interrupt line. Setting any of these bits enables an interrupt generation. The corresponding status register bit generates an interrupt when the corresponding interrupt signal enable bit is set.

Offset: 0x038 (IRQSIGEN) Access: Read/Write

| | 0–2 | 3 | 4–6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | DMAE IEN | — | AC12E IEN | — | DEBE IEN | DCE IEN | DTOE IEN | CIE IEN | CEBE IEN | CCE IEN | CTOE IEN |
| Reset | All zeros | | | | | | | | | | | |

| | 16–22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | CINT IEN | CRM IEN | CINS IEN | BRR IEN | BWR IEN | DINT IEN | BGE IEN | TC IEN | CC IEN |
| Reset | All zeros | | | | | | | | | |

**Figure 20-14. Interrupt Signal Enable Register (IRQSIGEN)**

**Table 20-19. IRQSIGEN Field Descriptions**

| Field | Description |
|---|---|
| 0–2 | Reserved |
| 3<br>DMAEIEN | DMA error interrupt enable<br>0 Masked<br>1 Enabled |
| 4–6 | Reserved |
| 7<br>AC12EIEN | Auto CMD12 error interrupt enable<br>0 Masked<br>1 Enabled |
| 8 | Reserved |
| 9<br>DEBEIEN | Data end bit error interrupt enable<br>0 Masked<br>1 Enabled |
| 10<br>DCEIEN | Data CRC error interrupt enable<br>0 Masked<br>1 Enabled |
| 11<br>DTOEIEN | Data timeout error interrupt enable<br>0 Masked<br>1 Enabled |
| 12<br>CIEIEN | Command index error interrupt enable<br>0 Masked<br>1 Enabled |

### Table 20-19. IRQSIGEN Field Descriptions (continued)

| Field | Description |
|---|---|
| 13<br>CEBEIEN | Command end bit error interrupt enable<br>0 Masked<br>1 Enabled |
| 14<br>CCEIEN | Command CRC error interrupt enable<br>0 Masked<br>1 Enabled |
| 15<br>CTOEIEN | Command timeout error interrupt enable<br>0 Masked<br>1 Enabled |
| 16–22 | Reserved |
| 23<br>CINTIEN | Card interrupt signal enable<br>0 Masked<br>1 Enabled |
| 24<br>CRMIEN | Card removal interrupt enable<br>0 Masked<br>1 Enabled |
| 25<br>CINSIEN | Card insertion interrupt enable<br>0 Masked<br>1 Enabled |
| 26<br>BRRIEN | Buffer read ready interrupt enable<br>0 Masked<br>1 Enabled |
| 27<br>BWRIEN | Buffer write ready interrupt enable<br>0 Masked<br>1 Enabled |
| 28<br>DINTIEN | DMA interrupt enable<br>0 Masked<br>1 Enabled |
| 29<br>BGEIEN | Block gap event interrupt enable<br>0 Masked<br>1 Enabled |
| 30<br>TCIEN | Transfer complete interrupt enable<br>0 Masked<br>1 Enabled |
| 31<br>CCIEN | Command complete interrupt enable<br>0 Masked<br>1 Enabled |

### 20.4.13 Auto CMD12 Error Status Register (AUTOC12ERR)

When IRQSTAT[AC12E] is set, the host driver checks this register to identify what kind of error Auto CMD12 indicated. This register is valid only when IRQSTAT[AC12E] is set.

Offset: 0x03C (AUTOC12ERR) Access: Read

| | 0 | 23 | 24 | 25 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|
| R | — | | CNIB AC12E | — | AC12 IE | AC12 CE | AC12 EBE | AC12 TOE | AC12 NE |
| W | | | | | | | | | |
| Reset | All zeros | | | | | | | | |

**Figure 20-15. Auto CMD12 Error Status Register (AUTOC12ERR)**

**Table 20-20. AUTOC12ERR Field Descriptions**

| Field | Description |
|---|---|
| 0–23 | Reserved |
| 24<br>CNIBAC12E | Command not issued by Auto CMD12 error. This bit is set when CMD_wo_DAT is not executed due to an Auto CMD12 error (D04–D01).<br>0 No error<br>1 Not Issued |
| 25–26 | Reserved |
| 27<br>AC12IE | Auto CMD12 index error. Occurs if the command index error occurs in response to a command.<br>0 No error<br>1 Error, the CMD index in response is not CMD12 |
| 28<br>AC12CE | Auto CMD12 CRC error. Occurs when detecting a CRC error in the command response.<br>0 No CRC error<br>1 CRC error met in Auto CMD12 response |
| 29<br>AC12EBE | Auto CMD12 end bit error. Occurs when detecting that the end bit of command response is 0 when it should be 1.<br>0 No error<br>1 End bit error generated |
| 30<br>AC12TOE | Auto CMD12 timeout error. Occurs if no response is returned within 64 SDHC_CLK cycles from the end bit of the command. If this bit is set, the other error status bits (2–4) are meaningless.<br>0 No error<br>1 Time out |
| 31<br>AC12NE | Auto CMD12 not executed. If a memory multiple block data transfer is not started due to command error, this bit is not set because it is not necessary to issue Auto CMD12. Setting this bit means eSDHC cannot issue Auto CMD12 to stop the memory multiple block data transfer due to some error. If this bit is set, the other error status bits (1–4) are meaningless.<br>0 Executed<br>1 Not executed |

Table 20-21. Relationship Between Command CRC Error and Command Timeout Error for Auto CMD12

| Auto CMD12 CRC Error | Auto CMD12 Timeout Error | Types of Error |
|---|---|---|
| 0 | 0 | No error |
| 0 | 1 | Response timeout error |
| 1 | 0 | Response CRC error |
| 1 | 1 | SDHC_CMD line conflict |

There are three scenarios when AUTOC12ERR can be changed:

1. When eSDHC is going to issue Auto CMD12
   - Set AC12NE if Auto CMD12 cannot be issued due to an error in the previous command.
   - Clear AC12NE if Auto CMD12 is issued.
2. At the end bit of an Auto CMD12 response
   - Check received responses by checking the error bits 1–4.
   - Set if error is detected.
   - Clear if error is not detected.
3. Before reading AUTOC12ERR[CNIBAC12E]
   - Set CNIBAC12E if there is a command that cannot be issued
   - Clear CNIBAC12E if there is no command to issue

The timing of generating the Auto CMD12 error and writing to the command register is asynchronous. The command may be blocked by any Auto CMD12 error causing CNIBAC12E to be set. Therefore, it is suggested to read this register only when IRQSTAT[AC12E] is set. An Auto CMD12 error interrupt is generated when one of the error bits 0–4 is set1. The CNIBAC12E error bit does not generate an interrupt.

## 20.4.14 Host Controller Capabilities (HOSTCAPBLT)

The host controller capabilities provides the host driver with information specific to the eSDHC implementation. The value in this register does not change in a software reset, and any write to this register is ignored.

Offset: 0x040 (HOSTCAPBLT) Access: Read

| | 0 | | | | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | — | | | | | VS18 | VS30 | VS33 | SRS | DMAS | HSS | — | | MBL | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |

| | 16 | | | | | | | | | | | | | | | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | — | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Figure 20-16. Host Capabilities Register (HOSTCAPBLT)**

**Table 20-22. HOSTCAPBLT Field Descriptions**

| Field | Description |
|---|---|
| 0–4 | Reserved |
| 5<br>VS18 | Voltage support 1.8 V. This bit depends on the host system ability.<br>0 1.8 V not supported<br>1 1.8 V supported |
| 6<br>VS30 | Voltage support 3.0 V. This bit depends on the host system ability.<br>0 3.0 V not supported<br>1 3.0 V supported |
| 7<br>VS33 | Voltage support 3.3 V. This bit depends on the host system ability.<br>0 3.3 V not supported<br>1 3.3 V supported |
| 8<br>SRS | Suspend/resume support. Indicates if eSDHC supports suspend/resume functionality. If this bit is 0, the suspend and resume mechanism, as well as the read wait, are not supported and the host driver should not issue suspend or resume commands.<br>0 Not supported<br>1 Supported |
| 9<br>DMAS | DMA support. Indicates if eSDHC is capable of using internal DMA to transfer data between system memory and the data buffer directly.<br>0 DMA not supported<br>1 DMA supported |
| 10<br>HSS | High speed support. Indicates if the eSDHC supports high speed mode and the host system can supply the SD clock frequency from 25 to 50 MHz.<br>0 High speed supported<br>1 High speed supported |
| 11–12 | Reserved |

**Table 20-22. HOSTCAPBLT Field Descriptions (continued)**

| Field | Description |
|---|---|
| 13–15<br>MBL | Max block length. Indicates the maximum block size that the host driver can read and write to the buffer in the eSDHC. The buffer should transfer block size without wait cycles.<br>000 512 bytes<br>001 1024 bytes<br>010 2048 bytes<br>011 4096 bytes |
| 16–31 | Reserved |

## 20.4.15 Watermark Level Register (WML)

Both write and read watermark levels are configurable. The value can be any number from 1–128 words.

Offset: 0x044 (WML) Access: Read/Write

| | 0 … 7 | 8 … 15 | 16 … 23 | 24 … 31 |
|---|---|---|---|---|
| R<br>W | — | WR_WML | — | RD_WML |
| Reset | 0 0 0 0 0 0 0 0 | 0 0 0 1 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 1 0 0 0 0 |

**Figure 20-17. Watermark Level Register (WML)**

**Table 20-23. WML Field Descriptions**

| Field | Description |
|---|---|
| 0–7 | Reserved |
| 8–15<br>WR_WML | Write watermark level. Number of 32-bit words of watermark level in DMA write operation. Also, the number of words of write burst length.<br>**Note:** The minimum value is 0x02, which represents 2 words (8 bytes). |
| 16–23 | Reserved |
| 24–31<br>RD_WML | Read watermark level. Number of 32-bit words of watermark level in DMA read operation. Also, the number of words of read burst length.<br>**Note:** The minimum value for RD_WML is 0x02, which means 2 words (8 bytes), and the maximum value for RD_WML is 0x10, which means 16 words (64 bytes). Setting RD_WML to values outside this range results in non-predicted behavior. |

## 20.4.16 Force Event Register (FEVT)

The force event register is not a physically implemented register. Rather, it is an address to which the IRQSTAT register can be written if the corresponding bit of IRQSTATEN is set. Therefore, this register is a write-only register and writing zero has no effect. Writing 1 to this register sets the corresponding bit of IRQSTAT. Reading from this register always returns zeroes.

Forcing a card interrupt generates a short pulse on the SDHC_DAT[1] line, and the driver may treat this interrupt as normal. The interrupt service routine may skip polling the card-interrupt source as the interrupt is self-cleared.

Offset: 0x050 (FEVT) Access: Write

| | 0 | 1 2 | 3 | 4 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | 0 | — | 0 | — | 0 | — | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| W | FEVT CINT | | FEVT DMAE | | FEVT AC12E | | FEVT DEBE | FEVT DCE | FEVT DTOE | FEVT CIE | FEVT CEBE | FEVT CCE | FEVT CTOE |
| Reset | All zeros | | | | | | | | | | | | |

| | 16 23 | 24 | 25 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|
| R | — | 0 | — | 0 | 0 | 0 | 0 | 0 |
| W | | FEVTCNI BAC12E | | FEVTA C12IE | FEVTA C12EBE | FEVTA C12CE | FEVTA C12TOE | FEVTA C12NE |
| Reset | All zeros | | | | | | | |

**Figure 20-18. Force Event Register (FEVT)**

**Table 20-24. FEVT Field Descriptions**

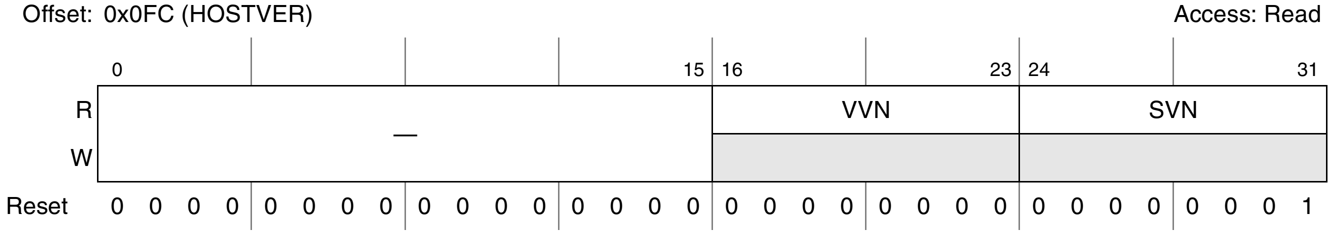
| Field | Description |
|---|---|
| 0<br>FEVTCINT | Force event card interrupt. Writing 1 to this bit generates a low-level short pulse on the internal SDHC_DAT[1] line, which imitates a self-clearing interrupt from the external card. If enabled, IRQSTAT[CINT] is set and the interrupt service routine may treat this interrupt as a normal interrupt from the external card. |
| 1–2 | Reserved |
| 3<br>FEVTDMAE | Force event DMA error. Forces IRQSTAT[DMAE] to set. |
| 4–6 | Reserved |
| 7<br>FEVTAC12E | Force event Auto CMD12 error. Forces IRQSTAT[AC12E] to set. |
| 8 | Reserved |
| 9<br>FEVTDEBE | Force event data end bit error. Forces IRQSTAT[DEBE] to set. |
| 10<br>FEVTDCE | Force event data CRC error. Forces IRQSTAT[DCE] to set. |
| 11<br>FEVTDTOE | Force event data time out error. Forces IRQSTAT[DTOE] to set. |
| 12<br>FEVTCIE | Force event command index error. Forces IRQSTAT[CCE] to set. |
| 13<br>FEVTCEBE | Force event command end bit error. Forces IRQSTAT[CEBE] to set. |
| 14<br>FEVTCCE | Force event command CRC error. Forces IRQSTAT[CCE] to set. |
| 15<br>FEVTCCE | Force event command time out error. Forces IRQSTAT[CTOE] to set. |

Table 20-24. FEVT Field Descriptions (continued)

| Field | Description |
|---|---|
| 16–23 | Reserved |
| 24<br>FEVTCNIBAC12E | Force event command not executed by Auto CMD12 error. Forces AUTOC12ERR[CNIBAC12E] to set. |
| 25–26 | Reserved |
| 27<br>FEVTAC12IE | Force event Auto CMD12 index error. Forces AUTOC12ERR[AC12IE] to set. |
| 28<br>FEVTAC12EBE | Force event Auto CMD12 end bit error. Forces AUTOC12ERR[AC12EBE] to set. |
| 29<br>FEVTAC12CE | Force event Auto CMD12 CRC error. Forces AUTOC12ERR[AC12CE] to set. |
| 30<br>FEVTAC12TOE | Force event Auto CMD12 time out error. Forces AUTOC12ERR[AC12TOE] to set. |
| 31<br>FEVTAC12NE | Force event Auto CMD12 not executed. Forces AUTOC12ERR[AC12NE] to set. |

## 20.4.17 Host Controller Version Register (HOSTVER)

The host controller version register contains the version for the vendor and the host controller. All the bits are read-only.



Offset: 0x0FC (HOSTVER) Access: Read

| | 0–15 | 16–23 | 24–31 |
|---|---|---|---|
| R | — | VVN | SVN |
| W | | | |
| Reset | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 1 |

Figure 20-19. Host Controller Version Register (HOSTVER)

Table 20-25. HOSTVER Field Descriptions

| Field | Description |
|---|---|
| 0–15 | Reserved |
| 16–23<br>VVN | Vendor version number. The host driver should not use this status. The upper and the lower 4-bits indicate the version.<br>0x00 Freescale eSDHC version 1.0<br>0x01 Freescale eSDHC version 2.0<br>others Reserved |
| 24–31<br>SVN | Specification version number. Indicates for the host controller specification version. The upper and the lower 4-bits indicate the version.<br>0x00 SD Host Specification Version 1.0<br>0x01 SD Host Specification Version 2.0, supports the test event register.<br>others Reserved |

## 20.4.18 DMA Control Register (DCR)

The DMA control register controls, shown in Figure 20-20, various settings that affect the system response to DMA operations.

Offset: 0x40C (DCR)

Access: Read/Write

|  | 0–15 |
|---|---|
| R<br>W | — |
| Reset | All zeros |

|  | 16–24 | 25 | 26–28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|
| R<br>W | — | SNOOP |  | RD_SAFE | RD_PFE | RD_PF_SIZE |
| Reset | All zeros |  |  |  |  |  |

**Figure 20-20. DMA Control Register (DCR)**

**Table 20-26. DCR Field Descriptions**

| Field | Description |
|---|---|
| 0–24 | Reserved. |
| 25<br>SNOOP | Snoop attribute.<br>0 DMA transactions are not snooped by the CPU data cache<br>1 DMA transactions are snooped by the CPU data cache |
| 26–28 | Reserved. |
| 29<br>RD_SAFE | Read safe. This bit should be set only if the target of a read-DMA operation is a well-behaved memory that is not affected by the read operation and returns the same data if read again from the same location. This means that unaligned reading operation can be rounded up to enable more efficient read operations.<br>0 It is not safe to read more bytes that were intended<br>1 It is safe to read more bytes that were intended |
| 30<br>RD_PFE | Read prefetch enable. This bit should be set if the target of read-DMA operation is a well-behaved memory that is not affected by the read operation and returns the same data if read again from the same location. This means that prefetch of data can be done by the internal bus units and it results in faster read completion.<br>0 It is not allowed to prefetch data on DMA read operation<br>1 It is allowed to prefetch data on DMA read operation |
| 31<br>RD_PF_SIZE | Read prefetch size. Determines the prefetch byte count to be used if RD_PFE is set.<br>0 64 bytes prefetch<br>1 32 bytes prefetch |

## 20.5 Functional Description

The following sections provide a brief functional description of the major system blocks, including the data buffer, DMA CCB interface, register bank, register bus interface, dual-port memory wrapper, data/command controller, clock and reset manager, and clock generator.

## 20.5.1 Data Buffer

The eSDHC uses one configurable data buffer so that data can be transferred between the internal system bus (register bus or CCB bus) and the SD card in an optimized manner to maximize throughput between the two clock domains (the IP peripheral clock and the master clock). See Figure 20-21 for an illustration of the buffer scheme.

The buffer is used as temporary storage for data being transferred between the host system and the card. The burst lengths for read and write are both configurable and can be any value between 1 and 128 words.

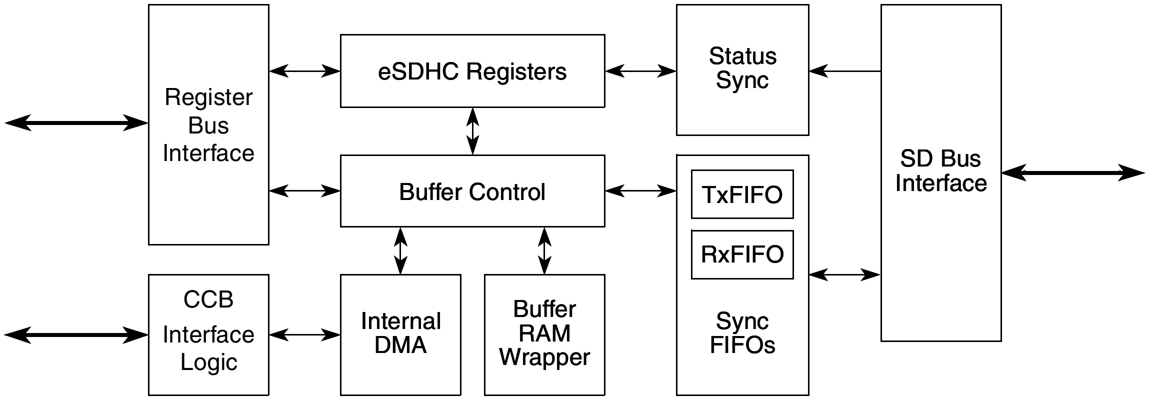


**Figure 20-21. eSDHC Buffer Scheme**

For a host read operation, when the amount of data exceeds the RD_WML value, the eSDHC sets PRSSTAT[BREN] and either:

- Issues a DMA request to inform the system to read the data
- Issues a DMA interrupt to inform the system to read the data
- When granted CCB access permission, the internal DMA burst-reads RD_WML number of words

Conversely, for a host write operation, when the amount of buffer spaces exceeds the WR_WML value, the eSDHC sets PRSSTAT[BWEN] and either:

- Issues a DMA request to inform the system to write data to the buffer
- Issues a DMA interrupt to inform the system to write data to the buffer
- When granted CCB access permission, the internal DMA burst-writess WR_WML number of words into the buffer

### 20.5.1.1 Write Operation Sequence

There are two ways to write data into the buffer when the user transfers data to the card:

- Processor core polling IRQSTAT[BWR] (interrupt or polling)
- Internal DMA

When the internal DMA is not used (XFERTYP[DMAEN] is not set when the command is sent), the eSDHC asserts an external DMA request when more than WML[WR_WML] number of empty buffer word slots are available and ready for receiving new data. At the same time, the eSDHC sets IRQSTAT[BWR]. The buffer write ready interrupt is generated if it is enabled by software.

When the internal DMA is used, the eSDHC does not inform the system before all the required number of bytes are transferredand no error is encountered. When an error occurs during the data transfer, the eSDHC aborts the data transfer and abandons the current block. The host driver should read the content of the DMA system address register to obtain the start address of abandoned data block. If the current data transfer is in multi-block mode, the eSDHC does not automatically send CMD12 even though XFERTYP[AC12EN] is set. Therefore, in this scenario, the host driver should send CMD12 and restart the write operation from that address. It is recommended that a software reset for data is applied before the transfer is restarted after error recovery.

The eSDHC does not start data transmission until the WML[WR_WML] number of words of data can be held in the buffer. If the buffer is empty and the host system does not write data in time, the eSDHC stops the SDHC_CLK to avoid a data buffer underrun situation.

#### 20.5.1.2 Read Operation Sequence

There are two ways to read data from the buffer when transferring data to the card:

- Processor core polling IRQSTAT[BRR] (interrupt or polling)
- Internal DMA

When the internal DMA is not used (XFERTYP[DMAEN] is not set when the command is sent), the eSDHC asserts a DMA request when more than WML[RD_WML] number of words are available and ready for the system to fetch the data. At the same time, the eSDHC sets the IRQSTAT[BRR] bit. The buffer read ready interrupt is generated if it is enabled by software.

When the internal DMA is used, the eSDHC does not inform the system before all the required number of bytes are transferred and no error is encountered. When an error occurs during the data transfer, the eSDHC aborts the data transfer and abandons the current block. The host driver should read the content of the DMA system address register to obtain the start address of abandoned data block. If the current data transfer is in multi-block mode, the eSDHC does not automatically send CMD12 even though XFERTYP[AC12EN] is set. Therefore, in this scenario, the host driver should send CMD12 and restart the read operation from that address. It is recommended that a software reset for data is applied before the transfer is restarted after error recovery.

The eSDHC does not start data transmission until the WML[RD_WML] number of words of data are in the buffer. If the buffer is full and the host system does not read the data in time, the eSDHC stops the SDHC_CLK to avoid a data buffer overrun situation.

#### 20.5.1.3 Data Buffer Size

To use the buffer in the most optimized way, the buffer size must be known. In the eSDHC the data buffer can hold up to 128 32-bit words, and the read and write watermark levels are each configurable from 1–128 words. The host driver may configure the values according to the system situation and requirements.

During multi-block data transfer, the maximum block length is 4096 bytes, which can satisfy all the requirements from MMC and SD cards. Any block length less than this value is also allowed. The only restriction is from the external card since it may not support such a large block or a partial block access that is not an integer multiple of 512 bytes.

## 20.5.2 DMA CCB Interface

The internal DMA implements a DMA engine and CCB master. When the internal DMA is enabled (XFERTYP[DMAEN] is set), the buffer interrupt status bits are still set if they are enabled. To avoid setting them, clear IRQSTATEN[BWRSEN, BRRSEN]. See Figure 20-22 for illustration of the DMA CCB interface block. The internal DMA must not be used to read (or write) data if the data will be written (or read) by the CPU or an external DMA through the DATPORT register.

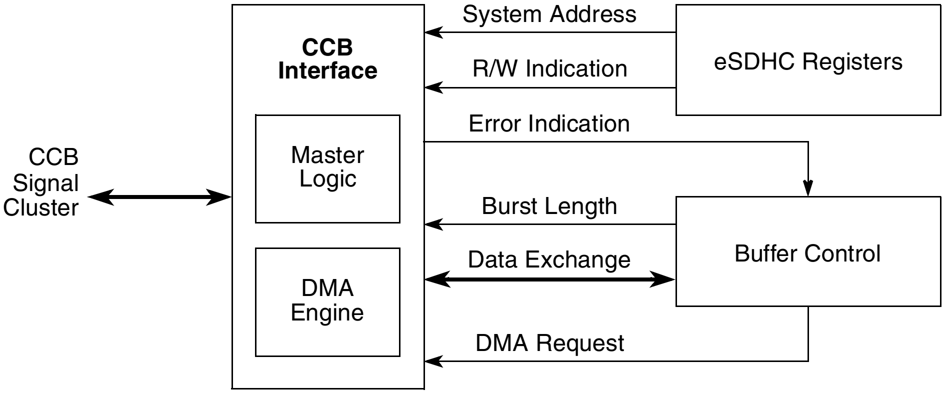


**Figure 20-22. DMA CCB Interface Block**

### 20.5.2.1 Internal DMA Request

If the watermark level is met in the data transfer and the internal DMA is enabled, the data buffer block sends a DMA request to this block. Meanwhile, the external DMA request is disabled. The delay of response from the internal DMA engine depends on the system bus loading and the priority assigned to eSDHC. The DMA engine does not respond to the request during its burst transfer, and is available as soon as the burst is over. The data buffer deasserts the request once an access on the buffer is made. Upon access to the buffer by the internal DMA, the data buffer updates its internal buffer pointer and when the watermark level is satisfied, another DMA request is sent.

The data transfer is in the block unit and the last watermark level is always set to the remaining number of words. For instance, for a multi-block data read with each block size of 31 bytes, the burst length is set at six words. After the first burst transfer, if there are more than seven bytes in the buffer, which might be partly some data of the next block, another DMA read request is sent because the remaining number of words to send for the current block is $(31 - 6 \times 4) \div 4 = 2$, and eSDHC reads two words out of the buffer, with seven valid bytes and one stuff byte automatically added by eSDHC.

### 20.5.2.2 DMA Burst Length

Just like the CPU polling access, the DMA burst length for the internal DMA engine does not a restriction other than the maximum size. The burst length for read or write can be 1–128 words. The actual burst length for the DMA depends on which is smaller: configured watermark level or the remaining words of current block.

Take the example in Section 20.5.2.1, “Internal DMA Request,” again. After six words are read, the burst length is two words to complete the 31-byte block. The burst length then changes back to six words to prepare for the next 31-byte block. The host driver writer may take this variable burst length into account.

It is also acceptable to configure the burst length as the divisor of block size so that each time the burst length is the same.

### 20.5.2.3 CCB Master Interface

It is possible that the internal DMA engine fails during the data transfer. When an error occurs, the DMA engine stops the transfer and goes to the idle state, while the internal data buffer stops working, too. IRQSTAT[DMAE] is set to inform the driver.

Once the IRQSTAT[DMAE] interrupt is received, software should send CMD12 to abort the current transfer and read DSADDR[DS_ADDR] to obtain the start address of the corrupted block. After the DMA error is fixed, the software should apply a data reset and restart the transfer from this address to recover the corrupted block.

## 20.5.3 SD Protocol Unit

The SD protocol unit deals with all SD protocol affairs and performs the following:

- Acts as the bridge between internal buffer and the SD bus
- Sends the command data and its argument serially
- Stores the serial response bit stream into corresponding registers
- Detects bus state on SDHC_DAT[0] line
- Asserts read wait signal
- Gates off SD clock when the buffer announces danger status
- Detects write-protect state
- And other functions

It consists of four submodules: SD transceiver, SD clock and monitor, command agent and data agent.

### 20.5.3.1 SD Transceiver

In the SD protocol unit, the transceiver is the main control module. It consists of a FSM and the control module, from which the control signals for all other three modules are generated.

### 20.5.3.2 SD Clock and Monitor

This module monitors the signal level on all eight data lines and the command lines, directly route the level values into the register bank for the driver to debug with.

The transceiver reports the card insertion state according to the $\overline{\text{SDHC\_CD}}$ state, or signal level on SDHC_DAT[3] line when PROCTL[D3CD] is set.

The module detects the SDHC_WP (write protect) line. With the information of SDHC_WP state, the register bank ignores the command accompanied by write operation, when the SD_WP switch is on.

If the internal data buffer is in danger and the SD clock must be gated off to avoid buffer over/underrun, this module asserts the gate of output SD clock to shut the clock off. When the buffer danger is eliminated when system access of the buffer catches up, the clock gate of this module is open and the SD clock is active again.

### 20.5.3.3 Command Agent

The command agent deals with the transactions on SDHC_CMD line. See Figure 20-23 for illustration of the structure for the command CRC shift register.

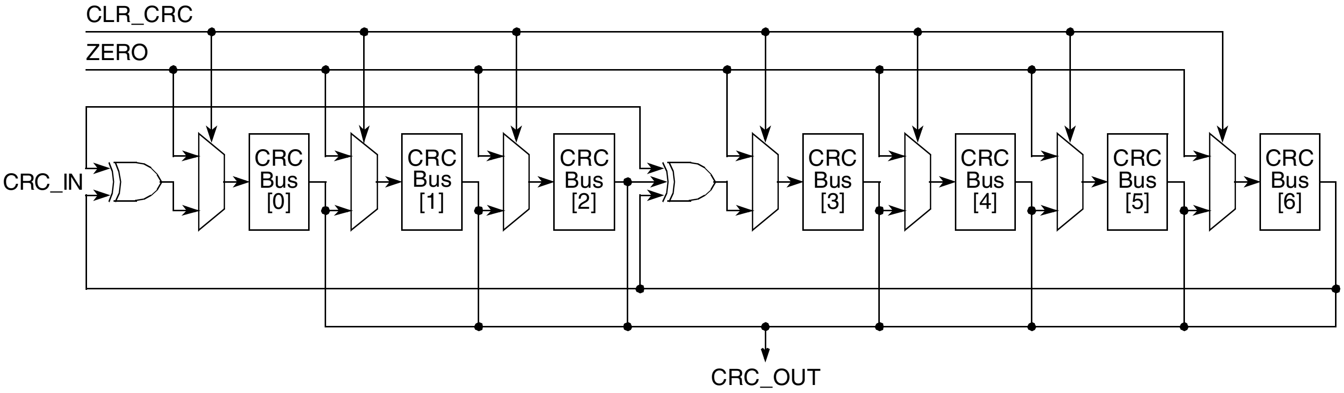


**Figure 20-23. Command CRC Shift Register**

The CRC polynomials for the SDHC_CMD are as follows:

Generator polynomial: $G(x) = x^7 + x^3 + 1$

$M(x) = (\text{first bit}) \times x^n + (\text{second bit}) \times x^{n-1} + ... + (\text{last bit}) \times x^0$

$CRC[6:0] = \text{Remainder } [(M(x) \times x^7) \div G(x)]$

### 20.5.3.4 Data Agent

The data agent handles the transactions on the eight data lines. Moreover, this module also detects the busy state from on SDHC_DAT[0] line, and generates read wait state by the request from the transceiver. The CRC polynomials for the SDHC_DAT are as follows:

Generator polynomial: $G(x) = x^{16} + x^{12} + x^5 + 1$

$M(x) = (\text{first bit}) \times x^n + (\text{second bit}) \times x^{n-1} + ... + (\text{last bit}) \times x^0$

$CRC[15:0] = \text{Remainder } [(M(x) \times x^{16}) \div G(x)]$

## 20.5.4 Clock & Reset Manager

This module controls all the reset signals within the eSDHC. There are four types of reset signals within eSDHC: hardware reset, software reset for all, software reset for data, and software reset for command. All these signals are fed into this module and stable signals are generated inside the module to reset all other modules.

This module also gates off all the inside signals. The module monitors the activities of all other modules, supplies the clocks for them, and when enabled, automatically gates off the corresponding clocks.

### 20.5.5 Clock Generator

The clock generator generates the SDHC_CLK by dividing the internal bus clock into two stages. Refer to Figure 20-24 for the structure of the divider, in which the term base represents the frequency of the internal bus clock (ccb_clk/2). Refer to SYSCTL[SDCLKFS] and SYSCTL[DVS] (see Section 20.4.9, "System Control Register (SYSCTL)") to select the divisor values.

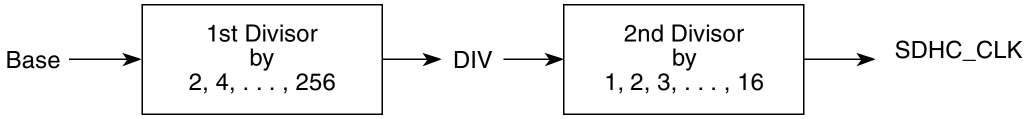


**Figure 20-24. Two Stages of Clock Divider**

The first stage is a prescaler. The frequency of clock output from this stage, DIV, can be base, base/2, base/4, ..., or base/256.

The second stage outputs the actual clock, SDHC_CLK, as the driving clock for all sub-modules of SD protocol unit, and the sync FIFOs in Figure 20-21 to synchronize with the data rate from the internal data buffer. It can be div, div/2, div/3,..., or div/16. Thus, the highest frequency of SDHC_CLK generated by the internal bus clock (ccb_clk/2) is base while the lowest frequency is base/4096.

### 20.5.6 Card Insertion and Removal Detection

The eSDHC uses the SDHC_DAT[3] pin or the $\overline{\text{SDHC\_CD}}$ pin to detect card insertion or removal. When SDHC_DAT[3] pin is used for card detection, user needs to pull-down this pad as a default state. When there is no card on the MMC/SD bus, the SDHC_DAT[3] is pulled to a low voltage level by default. When any card is inserted to or removed from the socket, the eSDHC detects the logic value changes on the SDHC_DAT[3] pin and generates an interrupt.

When SDHC_DAT[3] pin is not used for card detection, $\overline{\text{SDHC\_CD}}$ must be connected for card detection. It may be implemented by a GPIO. Whether SDHC_DAT[3] is configured for card detection or not, $\overline{\text{SDHC\_CD}}$ is always a reference for card detection, either SDHC_DAT[3] or $\overline{\text{SDHC\_CD}}$ reports card inserted, the eSDHC informs the host system that a card is inserted, and the interrupt is sent if it is enabled.

### 20.5.7 Power Management and Wake-Up Events

When there is no operation between eSDHC and the card through SD bus, you can completely disable the internal clocks in the chip level clock control module to save power. When you need to use the eSDHC to communicate with the card, it can enable the clock and start the operation. This can be done by clearing the SCCR[SDHCCM] bits.

In some circumstances, when the clocks to eSDHC are disabled, or when system is in low power mode, there are some events when you need to enable the clock and handle the event. These events are called wakeup interrupts. The eSDHC can generate these interrupts even there are no clocks enabled. The three interrupts which can be used as wake-up events are:

- Card removal interrupt
- Card insertion interrupt

These three wake-up events (or wake-up interrupts) can be also used to wake up the system from low-power modes.

**NOTE**

> To make the interrupt as wakeup event when all the clocks to eSDHC are disabled or when whole system is in low power mode, the corresponding wakeup enable bit need to be set. Refer to Section 20.4.8, “Protocol Control Register (PROCTL),” for more information on the eSDHC PROCTL register.

### 20.5.7.1 Setting Wake Up Events

For the eSDHC to respond to a wake up event, the software must set the respective wake up enable bit before the CPU enters sleep mode. Refer to Section 20.4.8, “Protocol Control Register (PROCTL),” for more information on the wakeup enable bits.

Before the software disables the host clock, it should ensure that all of the following conditions have been met:

- No read or write transfer is active
- Data and command lines are not active
- No interrupts are pending
- Internal data buffer is empty

## 20.6 Initialization/Application Information

All communication between system and cards are controlled by the host. The host sends commands of two types: broadcast and addressed (point-to-point) commands.

Broadcast commands are intended for all cards, such as GO_IDLE_STATE, SEND_OP_COND, ALL_SEND_CID, etc. In broadcast mode, all cards are in the open-drain mode to avoid bus contention. Refer to Section 20.6.5, “Commands for MMC/SD,” for the commands of bc and bcr categories.

After the broadcast command CMD3 is issued, the cards enter standby mode. Addressed type commands are used from this point. In this mode, the SDHC_CMD/SDHC_DAT I/O pads turn to push-pull mode, to have the driving capability for maximum frequency operation. Refer to Section 20.6.5, “Commands for MMC/SD,” for the commands of ac and adtc categories.

### 20.6.1 Command Send and Response Receive Basic Operation

Assuming data type WORD is an unsigned 32-bit integer, the below flow is a guideline for sending a command to the card(s):

```
send_command(cmd_index, cmd_arg, other requirements)
{
WORD wCmd; // 32-bit integer to make up the data to write into the XFERTYP register, it is
          // recommended to implement in a bit-field manner
wCmd = (<cmd_index> & 0x3f) << 24; // set the first 8 bits as ‘00’+<cmd_index>
set CMDTYP, DPSEL, CICEN, CCCEN, RSTTYP, and DTDSEL according to the command index;
         // XFERTYP register bits
if (internal DMA is used) wCmd |= 0x1;
if (multi-block transfer) {
         set XFERTYP[MSBSEL] bit;
         if (finite block number) {
                  set XFERTYP[BCEN] bit;
                  if (auto12 command is to use) set XFERTYP[AC12EN] bit;
         }
}
write_reg(CMDARG, <cmd_arg>); // configure the command argument
write_reg(XFERTYP, wCmd); // set XFERTYP register as wCmd value to issue the command
}
wait_for_response(cmd_index)
{
while (IRQSTAT[CC] is not set); // wait until command complete bit is set
read IRQSTAT register and check if any error bits about command are set;
if (any error bits are set) report error;
write 1 to clear IRQSTAT[CC] and all command error bits;
}
```

For the sake of simplicity, the function wait_for_response is implemented here by means of polling. For an effective and formal way, the response is usually checked after the command complete interrupt is received. By doing this, ensure the corresponding interrupt status bits are enabled.

For some scenarios, the response timeout is expected. For instance, after all cards respond to CMD3 and go to the standby state, no response to the host when CMD2 is sent. The host driver should manage false errors similar to this with caution.

## 20.6.2 Card Identification Mode

When a card is inserted to the socket or the card was reset by the host, the host needs to validate the operation voltage range, identify the cards, and request the cards to publish the relative card address (RCA) or to set the RCA for the MMCs.

### 20.6.2.1 Card Detect

See Figure 20-25 for a flow diagram showing the detection of MMC and SD cards using the eSDHC.

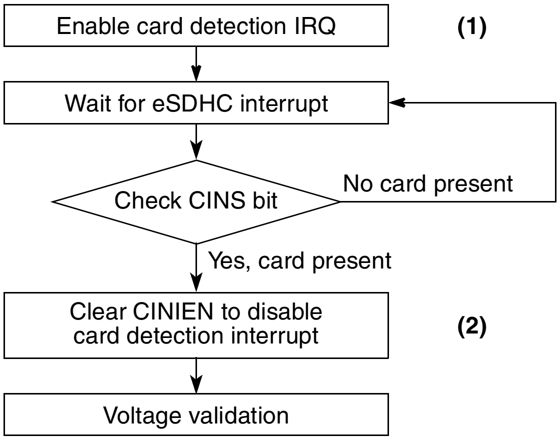


**Figure 20-25. Flow Diagram for Card Detection**

- Set IRQSIGEN[CINIEN] to enable card detection interrupt.
- When an interrupt from eSDHC is received, check IRQSTAT[CINS] to see if it is caused by card insertion.
- Clear the IRQSIGEN[CINIEN] to disable card detection interrupt and ignore all card insertion interrupt afterwards.

### 20.6.2.2 Reset

The host consists of three types of reset:

- Hardware reset (card and host) which is driven by POR (power on reset).
- Software reset (host only) is proceeded by the write operation on the SYSCTL[RSTD], SYSCTL[RSTC], or SYSCTL[RSTA] bits to reset the data part, command part, or all parts of the host controller, respectively.
- Card reset (card only). The command CMD0, GO_IDLE_STATE, is the software reset command for all types of MMCs and SD memory cards. This command sets each card into idle state regardless of the current card state. The cards are initialized with a default relative card address (RCA = 0x0000) and with a default driver stage register setting (lowest speed, highest driving current capability).

After the card is reset, the host needs to validate the voltage range of the card. See Figure 20-26 for the software flow to reset the eSDHC and card.

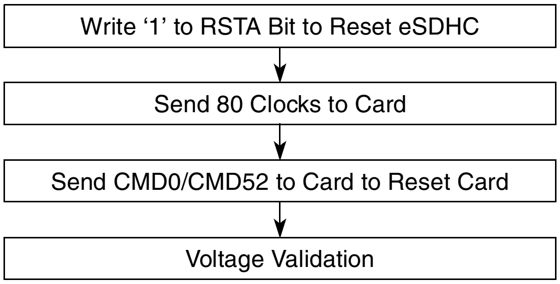

Write ‘1’ to RSTA Bit to Reset eSDHC
Send 80 Clocks to Card
Send CMD0/CMD52 to Card to Reset Card
Voltage Validation

**Figure 20-26. Flow Chart for Reset of eSDHC and SD I/O Card**

```
software_reset()
{
         set_bit(SYSCTL, RSTA);              // software reset the host
         set SYSCTL[DTOCV and SDCLKFS];      // get the SDHC_CLK of frequency around 400 KHz
         configure I/O pad;                  // set the voltage of external card to around 3.0 V
         poll PRSSTAT[CIHB and CDIHB];       // wait until both bits are cleared
         set_bit(SYSCTRL, INTIA);            // send 80 clock ticks for card to power-up
         send_command(CMD_GO_IDLE_STATE, <other parameters>); // reset the card with CMD0
         or send_command(CMD_IO_RW_DIRECT, <other parameters>);
}
```

### 20.6.2.3 Voltage Validation

All cards should be able to establish communication with the host using any operation voltage in the maximum allowed voltage range specified in this standard. However, the supported minimum and maximum values for $V_{DD}$ are defined in the operation conditions register (OCR) and may not cover the whole range. Cards that store the CID (card identification) and CSD data in the preloaded memory are only able to communicate these information under data transfer $V_{DD}$ conditions. This means that if the host and card have different $V_{DD}$ ranges, the card is not able to complete the identification cycle, nor is it able to send CSD data.

Therefore, a special command is available:

- SEND_OP_CONT (CMD1 for MMC),
- SD_SEND_OP_CONT (ACMD41 for SD Memory)

The voltage validation procedure is designed to provide a mechanism to identify and reject cards which do not match the $V_{DD}$ range(s) desired by the host. This is accomplished by the host sending the desired $V_{DD}$ voltage window as the operand of this command. Cards that can not perform data transfer in the specified range must discontinue any further bus operations and enter the inactive state. By omitting the voltage range in the command, the host can query each card and determine the common voltage range before sending out-of-range cards into the inactive state. This query should be used if the host is able to select a common voltage range or if a notification should be sent to the system when a non-usable cards in the stack is detected.

### 20.6.2.4 Card Registry

Card registry on MMC and SD/SD Combo cards are different.

For the SD card, the identification process starts at a clock rate lower than 400 KHz and the power voltage higher than 2.7 V, as defined by the card specification. At this time, the SDHC_CMD line output drives are push-pull drivers instead of open-drain. After the bus is activated, the host requests the card to send their valid operation conditions. The response to ACMD41 is the operation condition register of the card. The same command should be sent to all of the new cards in the system. Incompatible cards are placed into the inactive state. The host then issues the command, ALL_SEND_CID (CMD2), to each card to get its CID. Cards that are currently unidentified (that is, in ready state), send their CID number as the response. After the CID is sent by the card, the card goes into the identification state.

The host then issues Send_Relative_Addr (CMD3), requesting the card to publish a new relative card address (RCA) that is shorter than CID. This RCA is used to address the card for future data transfer operations. Once the RCA is received, the card changes its state to the standby state. At this point, if the host wants the card to have an alternative RCA number, it may ask the card to publish a new number by sending another Send_Relative_Addr command to the card. The last published RCA is the actual RCA of the card.

The host repeats the identification process with CMD2 and CMD3 for each card in the system until the last CMD2 gets no response from any of the cards in system.

For MMC operation, the host starts the card identification process in open-drain mode with the identification clock rate lower than 400 KHz, the power voltage higher than 2.7 V. The open-drain driver stages on the SDHC_CMD line allow parallel card operation during card identification. After the bus is activated the host requests the cards to send their valid operation conditions (CMD1). The response to CMD1 is the wired-OR operation on the condition restrictions of all cards in the system. Incompatible cards are sent into inactive state. The host then issues the broadcast command All_Send_CID (CMD2), asking all cards for their unique CID number. All unidentified cards (the cards in ready state) simultaneously start sending their CID numbers serially, while bit-wise monitoring their outgoing bit stream. Those cards, whose outgoing CID bits do not match the corresponding bits on the command line in any one of the bit periods, stop sending their CID immediately and must wait for the next identification cycle. Since the CID is unique for each card, only one card can successfully send its full CID to the host. This card then goes into identification state. Thereafter, the host issues Set_Relative_Addr (CMD3) to assign to this card a relative card address (RCA). Once the RCA is received, the card state changes to the stand-by state, and the card does not react in further identification cycles, and its output driver switches from open-drain to push-pull. The host repeats the process, namely CMD2 and CMD3, until the host receives a time-out condition to recognize completion of the identification process.

## 20.6.3 Card Access

These sections describe the supported access modes with external cards.

### 20.6.3.1 Block Write

This section describes the process of writing data to external cards in block mode.

#### 20.6.3.1.1 Normal Write

During block write (CMD24–27), one or more blocks of data are transferred from the host to the card with a CRC appended to the end of each block by the host. If the CRC fails, the card should indicate the failure on the SDHC_DAT line (see below). The transferred data is discarded and not written, and all further transmitted blocks (in multi-block write mode) are ignored.

If the host uses partial blocks whose accumulated length is not block-aligned and block misalignment is not allowed (CSD parameter WRITE_BLK_MISALIGN is not set), the card detects the block misalignment error and aborts programming before the beginning of the first misaligned block. The card sets the ADDRESS_ERROR error bit in the status register, defined in the MMC/SD Specification, and then waits in the receive-data state for a stop command while ignoring all further data transfers. The write operation is also aborted if the host attempts to write over a write-protected area.

For MMC and SD cards, programming the CID and CSD registers does not require a previous block length setting. The transferred data is also CRC protected. If a part of the CSD or CID register is stored in the ROM, this unchangeable section must match the corresponding section of the receive buffer. If this match fails, then the card reports an error and does not change any register contents.

Some cards may require a long and unpredictable period of time to write a block of data. After receiving a block of data and completing the CRC check, the card begins writing. If its write buffer is full and unable to accept new data from a new WRITE_BLOCK command, the card holds the SDHC_DAT line low. The host may poll the status of the card with a SEND_STATUS command (CMD13) cards, at any time and the card responds with its status. The card status indicates whether the card can accept new data or if the write process is still in progress. The host may deselect the card by issuing CMD7 (to select a different card) to change the card into the standby state and release the SDHC_DAT line without interrupting the write operation. When re-selecting the card, it reactivates the busy indication by pulling SDHC_DAT low if programming is still in progress and the write buffer is unavailable.

For simplicity, the software flow described below incorporates the internal DMA, and the write operation is a multi-block write with Auto CMD12 enabled. For the other method (CPU polling status) and different transfer nature, the internal DMA part of the procedure should be removed and alternative steps inserted.

1. Check the card status and wait until the card is ready for data.
2. Set the card block length.
   — MMC/SD cards — use SET_BLOCKLEN (CMD16)
3. Set eSDHC BLKATTR[BLKSIZE] to the same as the block length set in the card in step 2.
4. Set eSDHC BLKATTR[BLKCNT] with the number of blocks to send.
5. Disable the buffer write ready interrupt, configure the DMA setting, and enable the eSDHC DMA when sending the command with data transfer. Set XFERTYP[AC12EN].
6. Wait for the transfer complete interrupt.
7. Check the status bit to see if a read CRC error or any other errors occurred between sending Auto CMD12 and receiving the response.

#### 20.6.3.1.2 Write with Pause

The write operation can be paused during the transfer. Instead of stopping the SDHC_CLK at any time to pause all the operations which is also inaccessible to the host driver, the driver can set PROCTL[SABGREQ] to pause the transfer between the data blocks. Since there is no timeout condition in a write operation during the data blocks, a write operation to the cards can be paused in this way and if line SDHC_DAT0 is not required to de-assert to release busy state, no suspend command is needed.

Similar to the flow described in Section 20.6.3.1.1, “Normal Write,” the write with pause is shown with the same type of write operations:

1. Check the card status and wait until card is ready for data.
2. Set the card block length.
   — MMC/SD cards — use SET_BLOCKLEN (CMD16)
3. Set the eSDHC BLKATTR[BLKSIZE] to the same as the block length set in the card in step 2.
4. Set eSDHC BLKATTR[BLKCNT] with the number of blocks to send.
5. Disable the buffer write ready interrupt, configure the DMA setting, and enable the eSDHC DMA when sending the command with data transfer. Set XFERTYP[AC12EN].
6. Set PROCTL[SABGREQ].
7. Wait for the transfer complete interrupt.
8. Clear PROCTL[SABGREQ].
9. Check the status bit to see if a read CRC error occurred.
10. Set PROCTL[CREQ] to continue the read operation.
11. Wait for the transfer complete interrupt.
12. Check the status bit to see if a read CRC error or any other errors occurred between sending Auto CMD12 and receiving the response.

The number of blocks left during the data transfer is accessible by reading the content of BLKATTR[BLKCNT]. Due to the data transfers and setting PROCTL[SABGREQ] are concurrent, along with the delay of register read and the register setting, the actual number of blocks left may not be the same as the value read earlier. The driver should read the value of BLKATTR[BLKCNT] after the transfer is paused and the transfer complete interrupt is received.

It is also possible that the transfer of the last block begins when the stop-at-block-gap request is sent to the buffer. In this case, the next block gap is the actual end of the transfer, and therefore, the request is ignored. The driver should treat this as a non-pause transfer and a common write operation.

When the write operation is paused, the data transfer inside the host system does not stop and the transfer remains active until the data buffer is full. The eSDHC reads the resume command as a normal command with a data transfer, and it is the driver’s responsibility to set all the relevant registers before the transfer is resumed. If there is only one block to send when the transfer is resumed, XFERTYP[MSBSEL, BCEN, AC12EN] are set. However, the eSDHC automatically sends CMD12 to mark the end of multi-block transfer.

### 20.6.3.2 Block Read

#### 20.6.3.2.1 Normal Read

For block reads, the basic unit of a data transfer is a block whose maximum size is stored in areas defined in corresponding card specifications. A CRC is appended to the end of each block, ensuring data transfer integrity. CMD17, CMD18, CMD53, and so on, can initiate a block read. After completing the transfer, the card returns to the transfer state.

For multi-block reads, data blocks are continuously transferred until a stop command is issued. If the host uses partial blocks whose accumulated length is not block aligned and block misalignment is not allowed, the card which does not support partial block length, should detect the block misalignment at the beginning of the first misaligned block and report the error, depending on its card type.

For simplicity, the software flow described below incorporates the internal DMA, and the read operation is a multi-block read with Auto CMD12 enabled. For the other method (CPU polling status) and different transfer nature, the internal DMA part should be removed and the alternative steps are straightforward.

1. Check the card status and wait until the card is ready for data.
2. Set the card block length.
   — MMC/SD cards — use SET_BLOCKLEN (CMD16)
3. Set eSDHC BLKATTR[BLKSIZE] to the same as the block length set in the card in step 2.
4. Set eSDHC BLKATTR[BLKCNT] with the number of blocks to send.
5. Disable the buffer read ready interrupt, configure the DMA setting, and enable the eSDHC DMA when sending the command with data transfer. Set XFERTYP[AC12EN].
6. Wait for the transfer complete interrupt.
7. Check the status bit to see if a read CRC error or any other errors occurred between sending Auto CMD12 and receiving the response.

#### 20.6.3.2.2 Read with Pause

In general, the read operation is not able to pause.

Similar to the flow described in Section 20.6.3.2.1, “Normal Read,” the read with pause is shown with the same type of read operations:

1. Set PROCTL[RWCTL].
2. Check the card status and wait until the card is ready for data.
3. Set the card block length.
   — MMC/SD cards — use SET_BLOCKLEN (CMD16)
4. Set eSDHC BLKATTR[BLKSIZE] to the same as the block length set in the card in Step 2.
5. Set eSDHC BLKATTR[BLKCNT] with the number of blocks to send.
6. Disable the buffer read ready interrupt, configure the DMA setting, and enable the eSDHC DMA when sending the command with data transfer. Set XFERTYP[AC12EN].
7. Set PROCTL[SABGREQ].
8. Wait for the transfer complete interrupt.

9. Clear PROCTL[SABGREQ].
10. Check the status bit to see if a read CRC error occurred.
11. Set PROCTL[CREQ] to continue the read operation.
12. Wait for the transfer complete interrupt.
13. Check the status bit to see if a read CRC error or any other errors occurred between sending Auto CMD12 and receiving the response.

Similar to the write operation, it is possible to meet the ending block of the transfer when paused. In this case, the eSDHC ignores the stop-at-block-gap request and treats it as a command read operation.

Unlike the write operation, there is no remaining data inside the buffer when the transfer is paused. All data received before the pause is transferred to the host system. Whether or not a suspend command is sent, the internal data buffer is not flushed.

If the suspend command is sent and the transfer is later resumed by means of the resume command, the eSDHC takes the command as a normal one accompanied with data transfer, and it is left for the driver to set all the relevant registers before the transfer is resumed. If there is only one block to send when the transfer is resumed, XFERTYP[MSBSEL, BCEN] and IRQSTT[AC12EN] are set. However, the eSDHC automatically sends CMD12 to mark the end of a multi-block transfer.

### 20.6.3.3 Transfer Error

#### 20.6.3.3.1 CRC Error

At the end of a block transfer, a write CRC status error or read CRC error may occur. For this type of error, the last block received should be discarded because the integrity of the data block is not guaranteed. It is recommended to discard the following data blocks and re-transfer the block from the corrupted one. For a multi-block transfer, the host driver should issue CMD12 to abort the current process and start the transfer by a new data command. In this scenario, even when the XFERTYP[AC12EN, BCEN] are set, the eSDHC does not automatically send CMD12 because the last block is not transferred. On the other hand, if it is within the last block that CRC error occurs, Auto CMD12 is sent by the eSDHC. In this case, the driver should resend or re-obtain the last block with a single block transfer.

#### 20.6.3.3.2 Internal DMA Error

During the data transfer with the internal DMA, if the DMA engine encounters an error on the platform bus, the DMA operation is aborted and a DMA error interrupt is sent to the host system. When acknowledged by such an interrupt, the driver should calculate the start address of the data block where the error occurred. The start address can be calculated by either of the following methods:

- Read the DSADDR[DSADDR] field. The error occurs during the previous burst. Therefore, by taking the block size, the previous burst length, and the start address of the next burst transfer into account, one can obtain the start address of the corrupted block.
- Read the BLKATTR[BLKCNT] field. The start address of the corrupted block can be calculated by the number of blocks left, the total number to transfer, the start address of transfer, and the size of each block. However, if BCEN is not set, the contents of the block attribute register does not change and this method does not work.

When a DMA error occurs, it is recommended to abort the current transfer by means of CMD12 (for multi-block transfer), apply a reset for data, and restart the transfer from the corrupted block to recover the error.

#### 20.6.3.3.3 Auto CMD12 Error

After the last block of a multi-block transfer is sent or received and XFERTYP[AC12EN] is set when the data transfer is initiated by the data command, the eSDHC automatically sends CMD12 to the card to stop the transfer. When an error occurs at this point, it is recommended that the host driver responds by:

1. Auto CMD12 response timeout. It is not certain whether the command has been accepted by the card or not. The driver should clear the Auto CMD12 error status bits and resend CMD12 until it is accepted by the card.
2. Auto CMD12 response CRC error. Since CMD12 has been received by the card, the card aborts the transfer. The driver may ignore the error and clear the error status bit.
3. Auto CMD12 conflict error or not sent. The command was not sent. Therefore, the driver should send CMD12 manually.

### 20.6.3.4 Card Interrupt

The external cards can inform the host controller through the use of special signals.

## 20.6.4 Switch Function

MMCs transferring data with a bus width other than one-bit wide is a new feature added to the MMC specification. The high-speed timing mode for all card devices is also newly-defined in recent various card specifications. To enable these new features, a type of switch command should be issued by the host driver.

For SD cards, the high-speed mode is queried and enabled by CMD6 (with the mnemonic symbol as SWICH_FUNC); for MMCs, the high-speed mode is queried by CMD8 and enabled by CMD6 (with the mnemonic symbol as SWITCH).

The 4-bit and 8-bit bus width of MMC is also enabled by the SWITCH command, but with a different argument.

These new functions can also be disabled by software reset , but such manner of restoring to normal mode is not recommended because a complete identification process is needed before the card is ready for data transfer.

For simplicity, the following flowcharts do not show a current capability check, which is recommended in the function switch process.

### 20.6.4.1 Query, Enable and Disable SD High Speed Mode

```
enable_sd_high_speed_mode(void)
{
        set BLKATTR[BLKCNT] to 1 (block), set BLKATTR[BLKSIZE] to 64 (bytes);
        send CMD6, with argument 0xFFFFF1 and read 64 bytes of data accompanying the R1
                response;
        wait data transfer done bit is set;
        check if the bit 401 of received 512 bit is set;
        if (bit 401 is ‘0’) report the SD card does not support high speed mode and return;
        send CMD6, with argument 0x80FFFFF1 and read 64 bytes of data accompanying the R1
                response;
        check if the bit field 379~376 is 0xF;
        if (the bit field is 0xF) report the function switch failed and return;
        change clock divisor value or configure the system clock feeding into eSDHC to generate
                the card_clk of around 50MHz;
        (data transactions like normal peers)
}
disable_sd_high_speed_mode(void)
{
        set BLKCNT field to 1 (block), set BLKSIZE field to 64 (bytes);
        send CMD6, with argument 0x80FFFFF0 and read 64 bytes of data accompanying the R1
                response;
        check if the bit field 379~376 is 0xF;
        if (the bit field is 0xF) report the function switch failed and return;
        change clock divisor value or configure the system clock feeding into eSDHC to generate
                the card_clk of the desired value below 25MHz;
        (data transactions like normal peers)
}
```

### 20.6.4.2 Query, Enable and Disable MMC High Speed Mode

```
enable_mmc_high_speed_mode(void)
{
        send CMD9 to get CSD value of MMC;
        check if the value of SPEC_VER field is 4 or above;
        if (SPEC_VER value is less than 4) report the MMC does not support high speed mode and
                return;
        set BLKCNT field to 1 (block), set BLKSIZE field to 512 (bytes);
        send CMD8 to get EXT_CSD value of MMC;
        extract the value of CARD_TYPE field to check the ‘high speed mode’ in this MMC is
                26MHz or 52MHz;
        send CMD6 with argument 0x1B90100;
        send CMD13 to wait card ready (busy line released);
        send CMD8 to get EXT_CSD value of MMC;
        check if HS_TIMING byte (byte number 185) is 1;
        if (HS_TIMING is not 1) report MMC switching to high speed mode failed and return;
        change clock divisor value or configure the system clock feeding into eSDHC to generate
                the card_clk of around 26MHz or 52MHz according to the CARD_TYPE;
        (data transactions like normal peers)
}

disable_mmc_high_speed_mode(void)
{
        send CMD6 with argument 0x2B90100;
        set BLKCNT field to 1 (block), set BLKSIZE field to 512 (bytes);
        send CMD8 to get EXT_CSD value of MMC;
        check if HS_TIMING byte (byte number 185) is 0;
        if (HS_TIMING is not 0) report the function switch failed and return;
        change clock divisor value or configure the system clock feeding into eSDHC to generate
                the card_clk of the desired value below 20MHz;
        (data transactions like normal peers)
}
```

### 20.6.4.3 Set MMC Bus Width

```
change_mmc_bus_width(void)
{
         send CMD9 to get CSD value of MMC;
         check if the value of SPEC_VER field is 4 or above;
         if (SPEC_VER value is less than 4) report the MMC does not support multiple bit width
                  and return;
         send CMD6 with argument 0x3B70x00; (8-bit, x=2; 4-bit, x=1; 1-bit, x=0)
         send CMD13 to wait card ready (busy line released);
         (data transactions like normal peers)
}
```

## 20.6.5 Commands for MMC/SD

See Table 20-27 for the list of commands for the MMC/SD cards. Refer to the corresponding specifications for details about the command information.

Four kinds of commands control the MMC:

1. Broadcast commands (bc)—no response
2. Broadcast commands with response (bcr)—response from all cards simultaneously
3. Addressed (point-to-point) commands (ac)—no data transfer on SDHC_DAT
4. Addressed (point-to-point) data transfer commands (ADTC)

**Table 20-27. Commands for MMC/SD**

| CMD INDEX | Type | Argument | Resp | Abbreviation | Description[1] |
|---|---|---|---|---|---|
| CMD0 | bc | [31:0] stuff bits | — | GO_IDLE_STATE | Resets all MMC and SD memory cards to idle state. |
| CMD1 | bcr | [31:0] OCR without busy | R3 | SEND_OP_COND | Asks all MMCs and SD memory cards in idle state to send their operation conditions register contents in the response on the SDHC_CMD line. |
| CMD2 | bcr | [31:0] stuff bits | R2 | ALL_SEND_CID | Asks all cards to send their CID numbers on the SDHC_CMD line. |
| CMD3(1) | ac | [31:6] RCA<br>[15:0] stuff bits | R1 | SET/SEND_RELATIVE_ADDR | Assigns relative address to the card. |
| CMD4 | bc | [31:0] DSR<br>[15:0] stuff bits | — | SET_DSR | Programs the DSR of all cards. |
| CMD6(2) | adtc | [31] Mode<br>0: Check function<br>1: Switch function<br>[30:8] Reserved for function groups 6 ~ 3 (All 0 or 0xFFFF)<br>[7:4] Function group1 for command system<br>[3:0] Function group2 for access mode | R1 | SWITCH_FUNC | Checks switch ability (mode 0) and switch card function (mode 1). Refer to SD Physical Specification version 1.1 for details. |

Table 20-27. Commands for MMC/SD (continued)

| CMD INDEX | Type | Argument | Resp | Abbreviation | Description[1] |
|---|---|---|---|---|---|
| CMD6[(3)] | ac | [31:26] Set to 0<br>[25:24] Access<br>[23:16] Index<br>[15:8] Value<br>[7:3] Set to 0<br>[2:0] Cmd Set | R1b | SWITCH | Switches the mode of operation of the selected card or modifies the EXT_CSD registers. Refer to the MultiMediaCard System Specification version 4.0 final draft 2 for details. |
| CMD7 | ac | [31:6] RCA<br>[15:0] stuff bits | R1b | SELECT/DESELECT_CARD | Command toggles a card between the stand-by and transfer states or between the programming and disconnect states. In both cases, the card is selected by its own relative address and gets deselected by any other address; address 0 deselects all. |
| CMD8 | adtc | [31:0] stuff bits | R1 | SEND_EXT_CSD | The card sends its EXT_CSD register as a block of data, with block size of 512 bytes. |
| CMD9 | ac | [31:6] RCA<br>[15:0] stuff bits | R2 | SEND_CSD | Addressed card sends its card-specific data (CSD) on the SDHC_CMD line. |
| CMD10 | ac | [31:6] RCA<br>[15:0] stuff bits | R2 | SEND_CID | Addressed card sends its card-identification (CID) on the SDHC_CMD line. |
| CMD11 | adtc | [31:0] data address | R1 | READ_DAT_UNTIL_STOP | Reads data stream from the card starting at the given address until STOP_TRANSMISSION is received. |
| CMD12 | ac | [31:0] stuff bits | R1b | STOP_TRANSMISSION | Forces the card to stop transmission. |
| CMD13 | ac | [31:6] RCA<br>[15:0] stuff bits | R1 | SEND_STATUS | Addressed card sends its status register. |
| CMD14 | Reserved | | | | |
| CMD15 | ac | [31:6] RCA<br>[15:0] stuff bits | — | GO_INACTIVE_STATE | Sets the card to inactive state in order to protect the card stack against communication breakdowns. |
| CMD16 | ac | [31:0] block length | R1 | SET_BLOCKLEN | Sets the block length (in bytes) for all following block commands (read and write). Default block length is specified in the CSD. |
| CMD17 | adtc | [31:0] data address | R1 | READ_SINGLE_BLOCK | Reads a block of the size selected by the SET_BLOCKLEN command. |
| CMD18 | adtc | [31:0] data address | R1 | READ_MULTIPLE_BLOCK | Continuously transfers data blocks from card to host until interrupted by a stop command. |
| CMD19 | Reserved | | | | |

### Table 20-27. Commands for MMC/SD (continued)

| CMD INDEX | Type | Argument | Resp | Abbreviation | Description[1] |
|---|---|---|---|---|---|
| CMD20 | adtc | [31:0] data address | R1 | WRITE_DAT_UNTIL_STOP | Writes data stream from the host starting at the given address until the STOP_TRANSMISION command is received. |
| CMD21–23 | Reserved | | | | |
| CMD24 | adtc | [31:0] data address | R1 | WRITE_BLOCK | Writes a block of the size selected by the SET_BLOCKLEN command. |
| CMD25 | adtc | [31:0] data address | R1 | WRITE_MULTIPLE_BLOCK | Continuously writes blocks of data until the STOP_TRANSMISSION command is received. |
| CMD26 | adtc | [31:0] stuff bits | R1 | PROGRAM_CID | Programming of the card identification register. This command should be issued only once per card. The card contains hardware to prevent this operation after the first programming. Normally this command is reserved for the manufacturer. |
| CMD27 | adtc | [31:0] stuff bits | R1 | PROGRAM_CSD | Programming of the programmable bits of the CSD. |
| CMD28 | ac | [31:0] data address | R1b | SET_WRITE_PROT | If the card has write-protection features, this command sets the write protection bit of the addressed group. The properties of write protection are coded in the card-specific data (WP_GRP_SIZE). |
| CMD29 | ac | [31:0] data address | R1b | CLR_WRITE_PROT | If the card provides write-protection features, this command clears the write protection bit of the addressed group. |
| CMD30 | adtc | [31:0] write protect data address | R1 | SEND_WRITE_PROT | If the card provides write-protection features, this command asks the card to send the status of the write-protection bits. |
| CMD31 | Reserved | | | | |
| CMD32 | ac | [31:0] data address | R1 | TAG_SECTOR_START | Sets the address of the first sector of the erase group. |
| CMD33 | ac | [31:0] data address | R1 | TAG_SECTOR_END | Sets the address of the last write block of the continuous range to be erased. |
| CMD34 | ac | [31:0] data address | R1 | UNTAG_SECTOR | Removes one previously selected sector from the erase selection. |
| CMD35 | ac | [31:0] data address | R1 | TAG_ERASE_GROUP_STA RT | Sets the address of the first erase group within a range to be selected for erase. |

Table 20-27. Commands for MMC/SD (continued)

| CMD INDEX | Type | Argument | Resp | Abbreviation | Description[1] |
|---|---|---|---|---|---|
| CMD36 | ac | [31:0] data address | R1 | TAG_ERASE_GROUP_END | Sets the address of the last erase group within a continuous range to be selected for erase. |
| CMD37 | ac | [31:0] data address | R1 | UNTAG_ERASE_GROUP | Removes one previously selected erase group from the erase selection. |
| CMD38 | ac | [31:0] stuff bits | R1b | ERASE | Erase all previously selected sectors. |
| CMD39 | ac | [31:0] RCA<br>[15] register write flag<br>[14:8] register address<br>[7:0] register data | R4 | FAST_IO | Used to write and read 8-bit (register) data fields. The command address a card and a register and provides the data for writing if the write flag is set. The R4 response contains data read from the address register. This command accesses application dependent registers which are not defined in the MMC standard. |
| CMD40 | bcr | [31:0] stuff bits | R5 | GO_IRQ_STATE | Sets the system into interrupt mode. |
| CMD41 | Reserved | | | | |
| CDM42 | adtc | [31:0] stuff bits | R1b | LOCK_UNLOCK | Used to set/reset the password or lock/unlock the card. The size of the data block is set by the SET_BLOCK_LEN command. |
| CMD43–51 | Reserved | | | | |
| CMD52 | ac | [31:0] stuff bits | R5 | IO_RW_DIRECT | Access a single register within the total 128 Kbytes of register space in any I/O function. |
| CMD53 | ac | [31:0] stuff bits | R5 | IO_RW_EXTENDED | Access a multiple I/O register with a single command, it allows the reading or writing of a large number of I/O registers. |
| CMD54 | Reserved | | | | |
| CMD55 | ac | [31:16] RCA<br>[15:0] stuff bits | R1 | APP_CMD | Indicates to the card that the next command is an application specific command rather that a standard command. |
| CMD56 | adtc | [31:1] stuff bits<br>[0]: RD/WR | R1b | GEN_CMD | Used either to transfer a data block to the card or to get a data block from the card for general-purpose or application-specific commands. The size of the data block is set by the SET_BLOCK_LEN command. |
| CMD57–63 | Reserved | | | | |
| ACMDs should be preceded with the APP_CMD command<br>(Commands listed below are for SD cards only. Other SD commands not listed below are not supported by this module) | | | | | |

Table 20-27. Commands for MMC/SD (continued)

| CMD INDEX | Type | Argument | Resp | Abbreviation | Description[1] |
|---|---|---|---|---|---|
| ACMD6 | ac | [31:2] stuff bits<br>[1:0] bus width | R1 | SET_BUS_WIDTH | Defines the data bus width (00 = 1 bit or 10 = 4 bit bus) to be used for data transfer. The allowed data bus widths are given in DCR register. |
| ACMD13 | adtc | [31:0] stuff bits | R1 | SD_STATUS | Send the SD memory card status. |
| ACMD22 | adtc | [31:0] stuff bits | R1 | SEND_NUM_WR_ SECTORS | Send the number of the written (without errors) sectors. Responds with 32 bit + CRC data block. |
| ACMD23 | ac | [31:23] stuff bits<br>[22:0] number of blocks | R1 | SET_WR_BLK_ERASE_ COUNT | — |
| ACMD41 | bcr | [31:0] OCR | R3 | SD_APP_OP_COND | Asks the accessed card to send its operating condition register (OCR) content in the response on the SDHC_CMD line. |
| ACMD42 | ac | — | R1 | SET_CLR_CARD_DETECT | — |
| ACMD51 | adtc | [31:0] stuff bits | R1 | SEND_SCR | Reads the SD Configuration Register (SCR) |

[1] Registers mentioned in this table are SD card registers.

## NOTE

- CMD3 differs for MMC and SD cards
  For MMC cards, CMD3 is referred to as SET_RELATIVE_ADDR and has a response type R1
  For SD cards, CMD3 is referred to as SEND_RELATIVE_ADDR and has a response type R6, with RCA inside
- CMD6 differs completely between high-speed MMC cards and high-speed SD cards. Command SWITCH_FUNC is used for high speed SD cards.
- Command SWITCH is for high-speed MMC cards. The index field can contain any value from 0–255, but only values 0–191 are valid. If the index value is in the 192–255 range, the card does not perform any modification and the status bit EXT_CSD[SWITCH_ERROR] is set. The access bits are shown in Table 20-28:

Table 20-28. EXT_CSD Access Modes

| Bits | Access Name | Operation |
|---|---|---|
| 00 | Command set | The command set is changed according to the command set field of the argument |
| 01 | Set bits | The bits in the pointed byte are set, according to the set bits in the value field. |
| 10 | Clear bits | The bits in the pointed byte are cleared, according to the set bits in the value field. |
| 11 | Write byte | The value field is written into the pointed byte. |

### 20.6.6 Software Restrictions

When polling read or write, once the software begins a buffer read or write, it must access exactly the number of times as set in the watermark level register, as if a DMA burst occurred.

When the internal DMA is not enabled and a write transaction is in operation, DATPORT (described in Section 20.4.6, “Buffer Data Port Register (DATPORT)”) must not be read. DATPORT also must not be used to read (or write) data by the CPU or external DMA if the data will be written (or read) by the eSDHC internal DMA.

# Chapter 21
# Universal Serial Bus Interfaces

This chapter describes the universal serial bus (USB) interfaces of the device. The USB interface implements many industry standards. However, it is beyond the scope of this document to document the intricacies of these standards. Instead, it is left to the reader to refer to the governing specifications.

The following documents are available from the USB Implementers Forum web page at http://www.usb.org/developers/docs/.

- *Universal Serial Bus Revision 2.0 Specification*

The following documents are available from the Intel USB Specifications web page at http://www.intel.com/technology/usb/spec.htm.

- *Enhanced Host Controller Interface (EHCI) Specification for Universal Serial Bus, Revision 1.0*

The following documents are available from the ULPI web page at http://www.ulpi.org/

- *UTMI+ Specification, Revision 1.0*
- *UTMI Low Pin-Count Interface (ULPI) Specification, Revision 1.0*

## 21.1 Introduction

The MPC8536E has three dual-role (DR) USB interfaces (host or device). Figure 21-1 is a block diagram of one of the USB interfaces of the MPC8536E.

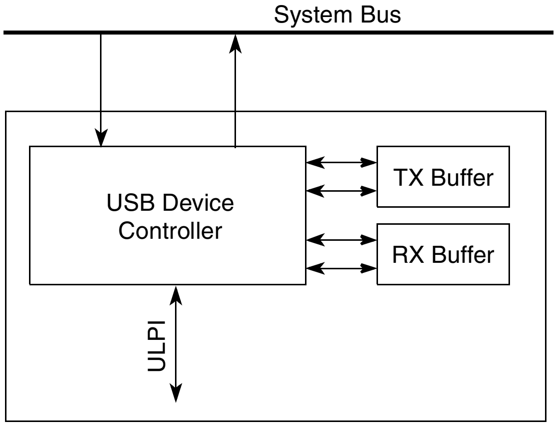


**Figure 21-1. USB Interface Block Diagram**

### 21.1.1 Overview

The USB DR module is a USB 2.0-compliant serial interface engine for implementing a USB interface. The registers and data structures are based on the *Enhanced Host Controller Interface Specification for Universal Serial Bus* (EHCI) from Intel Corporation. Each DR module is either a device or host controller.

The DR module supports the required signaling for UTMI low pin count interface (ULPI) transceivers (PHYs). The PHY interfacing to the ULPI is an external PHY.

The DR module contains a chaining DMA (direct memory access) engine that reduces the interrupt load on the application processor and reduces the total system bus bandwidth that must be dedicated to servicing the USB interface requirements.

### 21.1.2 Features

The USB DR module includes the following features:

- Complies with USB specification rev 2.0
- Supports operation as a standalone USB host controller
  - Supports enhanced host controller interface (EHCI)
- Supports high-speed (480 Mbps), full-speed (12 Mbps), and low-speed (1.5 Mbps) operation. Low speed is only supported in host mode.
- Supports external PHY with ULPI (UTMI + low-pin interface)
- Supports operation as a standalone USB device
  - Supports one upstream facing port
  - Supports six bidirectional USB endpoints
- Host or device support

### 21.1.3 Modes of Operation

The USB DR module operates in two modes: host and device.

**NOTE**

Only high-speed and full-speed operations are supported in device mode.

## 21.2 External Signals

This section contains detailed descriptions of all the USB controller signals.

### 21.2.1 ULPI Interface

The ULPI (UTMI low pin count interface) is a reduced pin-count (12 signals) extension of the UTMI+ specification. Pin count is reduced by converting relatively static signals to register bits, and providing a bidirectional, generic data bus that carries USB and register data. This interface minimizes pin count requirements for external PHYs. Table 21-1 describes the signals for the ULPI interface.

**Table 21-1. ULPI Signal Descriptions**

| Signal | I/O | Description | |
|---|---|---|---|
| USB*n*_DIR | I | Direction. USB*n*_DIR controls the direction of the data bus. When the PHY has data to transfer to USB port, it drives USB*n*_DIR high to take ownership of the bus. When the PHY has no data to transfer it drives USB*n*_DIR low and monitors the bus for link activity. The PHY pulls USB*n*_DIR high whenever the interface cannot accept data from the link. | |
| | | **State Meaning** | Asserted—PHY has data to transfer to the link.<br>Negated—PHY has no data to transfer. |
| | | **Timing** | Synchronous to PHY_CLK. |
| USB*n*_NXT | I | Next data. The PHY asserts USB*n*_NXT to throttle the data. When USB port is sending data to the PHY, USB*n*_NXT indicates when the current byte has been accepted by the PHY. The USB port places the next byte on the data bus in the following clock cycle. When the PHY is sending data to USB port, USB*n*_NXT indicates when a new byte is available for USB port to consume. | |
| | | **State Meaning** | Asserted—PHY is ready to transfer byte.<br>Negated—PHY is not ready. |
| | | **Timing** | Synchronous to PHY_CLK. |
| USB*n*_STP | O | Stop. USB*n*_STP indicates the end of a transfer on the bus. | |
| | | **State Meaning** | Asserted—USB asserts this signal for 1 clock cycle to stop the data stream currently on the bus. If USB port is sending data to the PHY, USB*n*_STP indicates the last byte of data was previously on the bus. If the PHY is sending data to USB port, USB*n*_STP forces the PHY to end its transfer, negate USB*n*_DIR and relinquish control of the data bus to the USB port.<br>Negated—Indicates normal operation. |
| | | **Timing** | Synchronous to PHY_CLK. |
| USB*n*_PWRFAULT | I | Power fault. USB*n*_PWRFAULT indicates whether a power fault occurred on the USB port Vbus.<br>**Note:** USB*n*_PWRFAULT, only exists for USB1 and USB2, not for USB3. | |
| | | **State Meaning** | Asserted—Indicates that a Vbus fault occurred. Applications that support power switching must shut down Vbus power.<br>Negated—Indicates normal operation. |
| | | **Timing** | Synchronous to PHY_CLK. |
| USB*n*_PCTL0 | O | Port control 0. USB*n*_PCTL0 controls the port status indicator LED 0 when in host mode.<br>**Note:** USB*n*_PCTL0, only exists for USB1 and USB2, not for USB3. | |
| | | **State Meaning** | Asserted—LED on.<br>Negated—LED off. |
| | | **Timing** | Synchronous to PHY_CLK. |
| USB*n*_PCTL1 | O | Port control 1. USB*n*_PCTL1 controls the port status indicator LED 1 when in host mode.<br>**Note:** USB*n*_PCTL1, only exists for USB1 and USB2, not for USB3. | |
| | | **State Meaning** | Asserted—LED on.<br>Negated—LED off. |
| | | **Timing** | Synchronous to PHY_CLK. |

**Table 21-1. ULPI Signal Descriptions (continued)**

| Signal | I/O | Description | |
|---|---|---|---|
| USB*n*_D[7:0] | I/O | Data bit *n*. USB*n*_D*n* is bit *n* of the 8-bit (USB*n*_D7–USB*n*_D0), uni-directional data bus used to carry USB, register, and interrupt data between the PHY and the USB controller. | |
| | | **State Meaning** | Asserted—Data bit *n* is 1.<br>Negated—Data bit *n* is 0. |
| | | **Timing** | Synchronous to PHY_CLK. |

## 21.2.2 PHY Clocks

The USB*n*_CLK input provides the clocking signal for the ULPI PHY interface. The clock is 60 MHz. Detailed clock specifications are given in the appropriate hardware specifications document.

**NOTE**

A write to registers in the USB controller memory map may cause the system to hang if PORTSC[PHCD]=0 when no USB PHY clock is applied.

## 21.3 Memory Map/Register Definitions

This section provides the memory map and detailed descriptions of all USB interface registers.

Table 21-2 shows the memory mapped registers of the USB controllers and their offsets. It lists the offset, name, and a cross-reference to the complete description of each register. Note that the full register address is comprised of CCSRBAR together with the USB controller block base address and offset listed in Table 21-2. Undefined 4-byte address spaces within offset 0x000–0xFFF are reserved.

**Table 21-2. USB Interface Memory Map**

| **USB Controller 1—Block Base Address 0x2_2000**<br>**USB Controller 2—Block Base Address 0x2_3000**<br>**USB Controller 3—Block Base Address 0x2_B000** | | | | |
|---|---|---|---|---|
| **Offset** | **Register** | **Access** | **Reset** | **Section/Page** |
| **USB Controller 1 Registers** | | | | |
| 0x000–0x0FF | Reserved, should be cleared | — | — | — |
| 0x100 | CAPLENGTH—Capability register length | R | 0x40 | 21.3.1.1/2121-7 |
| 0x102 | HCIVERSION—Host interface version number | R | 0x0100 | 21.3.1.2/2121-7 |
| 0x104 | HCSPARAMS—Host crtl. structural parameters | R | 0x0111_0011 | 21.3.1.3/2121-7 |
| 0x108 | HCCPARAMS—Host crtl. capability parameters | R | 0x0000_0006 | 21.3.1.4/2121-8 |
| 0x120 | DCIVERSION—Device interface version number | R | 0x0001 | 21.3.1.5/2121-9 |
| 0x124 | DCCPARAMS—Device controller parameters | R | 0x0000_0186 | 21.3.1.6/2121-10 |
| 0x140 | USBCMD—USB command | Mixed | 0x0008_*n*B00 | 21.3.2.1/2121-11 |
| 0x144 | USBSTS—USB status | Mixed | 0x0000_00*n*0 | 21.3.2.2/2121-13 |
| 0x148 | USBINTR—USB interrupt enable | R/W | 0x0000_0000 | 21.3.2.3/2121-15 |

**Table 21-2. USB Interface Memory Map (continued)**

| USB Controller 1—Block Base Address 0x2_2000<br>USB Controller 2—Block Base Address 0x2_3000<br>USB Controller 3—Block Base Address 0x2_B000 | | | | |
|---|---|---|---|---|
| **Offset** | **Register** | **Access** | **Reset** | **Section/Page** |
| 0x14C | FRINDEX—USB frame index | R/W | 0x0000_*nnnn* | 21.3.2.4/2121-17 |
| 0x154 | PERIODICLISTBASE—Frame list base address | R/W | 0x*nnnn*_0000 | 21.3.2.6/2121-18 |
| | DEVICEADDR—USB device address | R/W | 0x0000_0000 | 21.3.2.7/2121-19 |
| 0x158 | ASYNCLISTADDR—Next asynchronous list addr (host mode)[2] | R/W | 0x0000_0000 | 21.3.2.8/2121-19 |
| | ENDPOINT ADDR—Address at endpoint list (device mode) | R/W | 0x0000_0000 | 21.3.2.9/2121-20 |
| 0x160 | BURSTSIZE—Programmable burst size | R/W | 0x0000_1010 | 21.3.2.10/2121-21 |
| 0x164 | TXFILLTUNING—Host TT transmit pre-buffer packet tuning | R/W | 0x0002_0000 | 21.3.2.11/2121-21 |
| 0x170 | ULPI VIEWPORT—ULPI Register Access | Mixed | 0x0*n*00_0000 | 21.3.2.12/2121-23 |
| 0x180 | CONFIGFLAG—Configured flag register | R | 0x0000_0001 | 21.3.2.13/2121-24 |
| 0x184 | PORTSC—Port status/control | Mixed | 0x9C00_000*n* | 21.3.2.14/2121-25 |
| 0x1A8 | USBMODE—USB device mode | R/W | 0x0000_0000 | 21.3.2.15/2121-29 |
| 0x1AC | ENDPTSETUPSTAT—Endpoint setup status | R/W | 0x0000_0000 | 21.3.2.16/2121-30 |
| 0x1B0 | ENDPOINTPRIME—Endpoint initialization | R/W | 0x0000_0000 | 21.3.2.17/2121-31 |
| 0x1B4 | ENDPTFLUSH—Endpoint de-initialize | R/W | 0x0000_0000 | 21.3.2.18/2121-32 |
| 0x1B8 | ENDPTSTATUS—Endpoint status | R | 0x0000_0000 | 21.3.2.19/2121-32 |
| 0x1BC | ENDPTCOMPLETE—Endpoint complete | w1c | 0x0000_0000 | 21.3.2.20/2121-33 |
| 0x1C0 | ENDPTCTRL0—Endpoint control 0 | Mixed | 0x0080_0080 | 21.3.2.21/2121-33 |
| 0x1C4 | ENDPTCTRL1—Endpoint control 1 | R/W | 0x0000_0000 | 21.3.2.22/2121-35 |
| 0x1C8 | ENDPTCTRL2—Endpoint control 2 | R/W | 0x0000_0000 | 21.3.2.22/2121-35 |
| 0x1CA | ENDPTCTRL3—Endpoint control 3 | R/W | 0x0000_0000 | 21.3.2.22/2121-35 |
| 0x1D0 | ENDPTCTRL4—Endpoint control 4 | R/W | 0x0000_0000 | 21.3.2.22/2121-35 |
| 0x1D4 | ENDPTCTRL5—Endpoint control 5 | R/W | 0x0000_0000 | 21.3.2.22/2121-35 |
| 0x400 | SNOOP1—Snoop 1 | R/W | 0x0000_0000 | 21.3.2.23/2121-36 |
| 0x404 | SNOOP2—Snoop 2 | R/W | 0x0000_0000 | 21.3.2.23/2121-36 |
| 0x408 | AGE_CNT_THRESH—Age count threshold | R/W | 0x0000_0000 | 21.3.2.24/2121-37 |
| 0x40C | PRI_CTRL—Priority control | R/W | 0x0000_0000 | 21.3.2.25/2121-38 |
| 0x410 | SI_CTRL—System interface control | R/W | 0x0000_0000 | 21.3.2.26/2121-39 |
| 0x500 | CONTROL—Control | R/W | 0x0000_0000 | 21.3.2.27/2121-39 |
| 0x504–0xFFF | Reserved, should be cleared | — | — | — |
| **USB Controller 2 Registers** | | | | |

**Table 21-2. USB Interface Memory Map (continued)**

<table>
<tr><th colspan="5">USB Controller 1—Block Base Address 0x2_2000<br>USB Controller 2—Block Base Address 0x2_3000<br>USB Controller 3—Block Base Address 0x2_B000</th></tr>
<tr><th>Offset</th><th>Register</th><th>Access</th><th>Reset</th><th>Section/Page</th></tr>
<tr><td>0x000–<br>0xFFC</td><td colspan="4">USB controller 2 registers<br><b>Note:</b> All registers defined for USB controller 1 are also defined for USB controller 2; the offsets of USB controller 2 registers are the same except they have a different block base address.</td></tr>
<tr><th colspan="5">USB Controller 3 Registers</th></tr>
<tr><td>0x000–<br>0xFFC</td><td colspan="4">USB controller 3 registers<br><b>Note:</b> All registers defined for USB controller 1 are also defined for USB controller 3; the offsets of USB controller 3 registers are the same except they have a different block base address.</td></tr>
</table>

The following sections provide details about the registers in the USB memory map.

**NOTE**

Memory may be viewed from either a big-endian or little-endian byte ordering perspective depending on the processor configuration. In big-endian mode, the most-significant byte of word 0 is located at address 0 and the least-significant byte of word 0 is located at address 3. In little-endian mode, the least-significant byte of word 0 is located at address 0 and the most-significant byte of word 0 is located at address 3. Within registers, bits are numbered within a word starting with bit 31 as the most-significant bit. By convention USB registers use little-endian byte ordering. In the USB module, these are the registers from offsets 0x00 to 0x1FF. The registers associated with the internal system interface (0x400 and above) use big-endian byte ordering.

## 21.3.1 Capability Registers

The capability registers specify the software limits, restrictions, and capabilities of the host/device controller implementation. Most of these registers are defined by the EHCI specification. Registers that are not defined by the EHCI specification are noted in their descriptions.

#### 21.3.1.1 Capability Registers Length (CAPLENGTH)

CAPLENGTH is used as an offset to add to the register base address to find the beginning of the operational register space, that is, the location of the USBCMD register. Figure 21-2 shows CAPLENGTH.

Offset 0x100 Access: Read-only

| | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|
| R | CAPLENGTH | | | | | | | |
| W | | | | | | | | |
| Reset | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

**Figure 21-2. Capability Registers Length (CAPLENGTH)**

Table 21-3 provides bit descriptions for the CAPLENGTH register.

**Table 21-3. CAPLENGTH Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 7–0 | CAPLENGTH | Capability registers length. Value is 0x40. |

#### 21.3.1.2 Host Controller Interface Version (HCIVERSION)

HCIVERSION contains a BCD encoding of the EHCI revision number supported by this host controller. The most-significant byte of the register represents a major revision and the least-significant byte is the minor revision. Figure 21-3 shows the HCIVERSION register.

Offset 0x102 Access: Read-only

| | 15 | | | | | | | | | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | HCIVERSION | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Figure 21-3. Host Controller Interface Version (HCIVERSION)**

Table 21-4 provides bit descriptions for the HCIVERSION register.

**Table 21-4. HCIVERSION Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 15–0 | — | EHCI revision number. Value is 0x0100 indicating version 1.0. |

#### 21.3.1.3 Host Controller Structural Parameters (HCSPARAMS)

HCSPARAMS contains structural parameters such as the number of downstream ports. Figure 21-4 shows the HCSPARAMS register.

Offset 0x102 Access: Read-only

| | 31 28 | 27 24 | 23 20 | 19 17 | 16 | 15 12 | 11 8 | 7 5 | 4 | 3 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| R | — | N_TT | N_PTT | — | PI | N_CC | N_PCC | — | PPC | N_PORTS |
| W | | | | | | | | | | |
| Reset | 0 0 0 0 | 0 0 0 1 | 0 0 0 1 | 0 0 0 | 1 | 0 0 0 0 | 0 0 0 0 | 0 0 0 | 1 | 0 0 0 1 |

**Figure 21-4. Host Controller Structural Parameters (HCSPARAMS)**

Table 21-5 provides bit descriptions for the HCSPARAMS register.

**Table 21-5. HCSPARAMS Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31–28 | — | Reserved, should be cleared. |
| 27–24 | N_TT | Number of transaction translators. This is a non-EHCI field. This field indicates the number of embedded transaction translators associated the module. This field is always 1. See Section 21.9.1, “Embedded Transaction Translator Function.” |
| 23–20 | N_PTT | Ports per transaction translator. This is a non-EHCI field. The number of ports assigned to each transaction translator. This is equal to N_PORTS. |
| 19–17 | — | Reserved, should be cleared. |
| 16 | PI | Port indicators. Indicates whether the ports support port indicator control. Always 1.<br>1 The port status and control registers include a R/W field for controlling the state of the port indicator. |
| 15–12 | N_CC | Number of companion controllers associated with the USB controller. Always 0. |
| 11–8 | N_PCC | Number ports per CC. This field indicates the number of ports supported per internal companion controller. This field is always 0. |
| 7–5 | — | Reserved, should be cleared. |
| 4 | PPC | Power port control. Indicates whether the host controller supports port power control. It is always 1.<br>1 Ports have power port switches. |
| 3–0 | N_PORTS | Number of ports. Number of physical downstream ports implemented for host applications. The value of this field determines how many port registers are addressable in the operational register. Always 0x1. |

### 21.3.1.4 Host Controller Capability Parameters (HCCPARAMS)

HCCPARAMS identifies multiple mode control (time-base bit functionality) addressing capability. Figure 21-5 shows the HCCPARAMS register.

Offset 0x108 Access: Read-only

| | 31 16 | 15 8 | 7 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| R | — | EECP | IST | — | ASP | PFL | ADC |
| W | | | | | | | |
| Reset | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 | 0 | 1 | 1 | 0 |

**Figure 21-5. Host Control Capability Parameters (HCCPARAMS)**

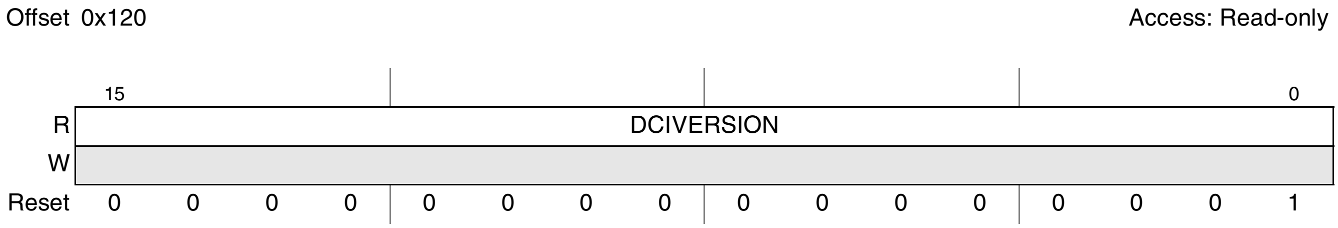
Table 21-6 provides bit descriptions for the HCCPARAMS register.

**Table 21-6. HCCPARAMS Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31–16 | — | Reserved, should be cleared. |
| 15–8 | EECP | EHCI extended capabilities pointer. Indicates the existence of a capabilities list. A value of 0x00 indicates no extended capabilities are implemented. A non-zero value in this register indicates the offset in PCI configuration space of the first EHCI extended capability. The pointer value must be 0x40 or greater if implemented to maintain the consistency of the PCI header defined for this class of device.<br>This field is always 0. |
| 7–4 | IST | Isochronous scheduling threshold. Indicates, relative to the current position of the executing host controller, where software can reliably update the isochronous schedule. When bit 7 is zero, the value of the least significant 3 bits indicates the number of microframes a host controller can hold a set of isochronous data structures (one or more) before flushing the state. When bit 7 is a one, then host software assumes the host controller may cache an isochronous data structure for an entire frame.<br>This field is always 0. |
| 3 | — | Reserved, should be cleared. |
| 2 | ASP | Asynchronous schedule park capability. Indicates whether the USB module supports the park feature for high-speed queue heads in the asynchronous schedule. The feature can be disabled or enabled and set to a specific level by using the asynchronous schedule park mode enable and asynchronous schedule park mode count fields in the USBCMD register.<br>This field is always 1 (park feature supported). |
| 1 | PFL | Programmable frame list flag. Indicates whether system software can specify and use a frame list length less that 1024 elements. Frame list size is configured via the USBCMD register frame list size field. The frame list must always be aligned on a 4K page boundary. This requirement ensures that the frame list is always physically contiguous.<br>This field is always 1. |
| 0 | ADC | 64-bit addressing capability. Always 0; 64-bit addressing is not supported.<br>0 Data structures use 32-bit address memory pointers |

## 21.3.1.5 Device Controller Interface Version (DCIVERSION)—Non-EHCI

This register is not defined in the EHCI specification. DCIVERSION is a two-byte register containing a BCD encoding of the device controller interface. The most-significant byte of the register represents a major revision and the least-significant byte is the minor revision. Figure 21-6 shows the DCIVERSION register.



Offset 0x120 Access: Read-only

| | 15 | | | | | | | | | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | DCIVERSION | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

**Figure 21-6. Device Interface Version (DCIVERSION)**

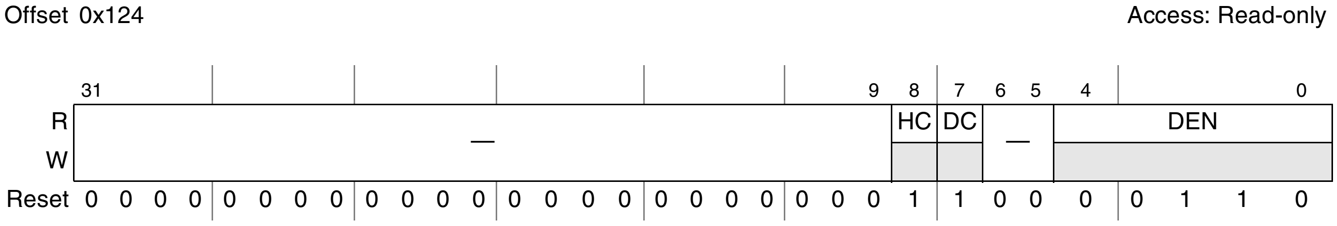
Table 21-7 provides bit descriptions for the DCIVERSION register.

**Table 21-7. DCIVERSION Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 15–0 | DCIVERSION | Device interface revision number. |

### 21.3.1.6 Device Controller Capability Parameters (DCCPARAMS)—Non-EHCI

This register is not defined in the EHCI specification. This register describes the overall host/device capability of the USB module. Figure 21-7 shows the DCCPARAMS register.



Offset 0x124 Access: Read-only

| | 31 … 9 | 8 | 7 | 6 5 | 4 … 0 |
|---|---|---|---|---|---|
| R | — | HC | DC | — | DEN |
| W | | | | | |
| Reset | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 1 | 1 | 0 0 | 0 0 1 1 0 |

**Figure 21-7. Device Control Capability Parameters (DCCPARAMS)**

Table 21-8 provides bit descriptions for the DCCPARAMS register.

**Table 21-8. DCCPARAMS Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31–9 | — | Reserved, should be cleared. |
| 8 | HC | Host capable. Always 1, indicating the USB controller can operate as an EHCI compatible USB 2.0 host. |
| 7 | DC | Device capable. Always 1, indicating the USB controller can operate as an USB 2.0 device.<br>1 Device capability<br>0 No device capability (host only) |
| 6–5 | — | Reserved, should be cleared. |
| 4–0 | DEN | Device endpoint number. Indicates the number of endpoints built into the device controller. Always 0x6. |

## 21.3.2 Operational Registers

The operational registers are comprised of dynamic control or status registers that may be read-only, read/write, or read/write-1-to-clear. The following sections define the operational registers.

### 21.3.2.1 USB Command Register (USBCMD)

The module executes the command indicated in this register.

Offset 0x140 Access: Mixed

| | 31 | | | | | | | 24 | 23 | | | | | | | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | — | | | | | | | | ITC | | | | | | | |
| W | | | | | | | | | ▨ | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | FS2 | ATDTW | SUTW | — | ASPE | — | ASP | | LR | IAA | ASE | PSE | FS1 | FS0 | RST | RS |
| W | | | | | | | | | ▨ | | | | | | | |
| Reset | 0 | *n* | *n* | 0 | 1 | 0 | 1 | 1 | 0 | *n* | 0 | 0 | 0 | 0 | 0 | 0 |

**Figure 21-8. USB Command Register (USBCMD)**

**Table 21-9. USBCMD Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31–24 | — | Reserved, should be cleared. |
| 23–16 | ITC | Interrupt threshold control. The system software uses this field to set the maximum rate at which the USB module will issue interrupts. ITC contains the maximum interrupt interval measured in microframes. Valid values are shown below.<br>0x00 Immediate (no threshold)<br>0x01 1 microframe<br>0x02 2 microframes<br>0x04 4 microframes<br>0x08 8 microframes<br>0x10 16 microframes<br>0x20 32 microframes<br>0x40 40 microframes |
| 15 | FS2 | See bits 3–2 below. This is a non-EHCI bit. |
| 14 | ATDTW | Add dTD TripWire. This is a non-EHCI bit. Used as a semaphore when a dTD is added to an active (primed) endpoint. This bit is set and cleared by software. This bit shall also be cleared by hardware when is state machine is hazard region where adding a dTD to a primed endpoint may go unrecognized. More information on the use of this bit is described in Section 21.9.2, "Device Operation." |
| 13 | SUTW | Setup tripwire. This is a non-EHCI bit. Used as a semaphore when the 8 bytes of setup data read extracted from a QH by the DCD. If the setup lockout mode is off (See USBMODE) then there exists a hazard when new setup data arrives and the DCD is copying setup from the QH for a previous setup packet. This bit is set and cleared by software and will be cleared by hardware when a hazard exists. More information on the use of this bit is described in Section 21.9.2, "Device Operation." |
| 12 | — | Reserved, should be cleared. |
| 11 | ASPE | Asynchronous schedule park mode enable. This bit defaults to a 1 and is R/W. Software uses this bit to enable or disable park mode.<br>0 Disabled<br>1 Enabled |
| 10 | — | Reserved, should be cleared. |

### Table 21-9. USBCMD Register Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 9–8 | ASP | Asynchronous schedule park mode count. This field defaults to 0x3H and is R/W. It contains a count of the number of successive transactions the host controller is allowed to execute from a high-speed queue head on the Asynchronous schedule before continuing traversal of the Asynchronous schedule. Valid values are 0x1H to 0x3H. Software must not write a zero to this field when ASPE is set as this will result in undefined behavior. |
| 7 | LR | Light host/device controller reset (OPTIONAL). Not implemented. Always 0. |
| 6 | IAA | Interrupt on async advance doorbell. Used as a doorbell by software to tell the USB controller to issue an interrupt the next time it advances asynchronous schedule. Software must write a 1 to this bit to ring the doorbell.<br>When the controller has evicted all appropriate cached schedule states, it sets USBSTS[AAI]. If USBINTR[AAE] is set, the host controller will assert an interrupt at the next interrupt threshold.<br>The controller clears this bit after it has set USBSTS[AAI]. Software should not set this bit when the asynchronous schedule is inactive. Doing so will yield undefined results.<br>This bit is only used in host mode. Setting this bit when the USB module is in device mode is selected will result in undefined results. |
| 5 | ASE | Asynchronous schedule enable. Controls whether the controller skips processing the asynchronous schedule. Only used in host mode.<br>0 Do not process the asynchronous schedule<br>1 Use the ASYNCLISTADDR register to access the asynchronous schedule. |
| 4 | PSE | Periodic schedule enable. Controls whether the controller skips processing the periodic schedule. Only used in host mode.<br>0 Do not process the periodic schedule.<br>1 Use the PERIODICLISTBASE register to access the periodic schedule. |
| 3–2 | FS | Frame list size. Together with bit 15 these bits make the FS[2:0] field. This field is read/write only if programmable frame list flag in the HCCPARAMS registers is set to 1. This field specifies the size of the frame list that controls which bits in FRINDEX should be used for the frame list current index. Only used in host mode. Note that values below 256 elements are not defined in the EHCI specification.<br>000 1024 elements (4096 bytes)<br>001 512 elements (2048 bytes)<br>010 256 elements (1024 bytes)<br>011 128 elements (512 bytes)<br>100 64 elements (256 bytes)<br>101 32 elements (128 bytes)<br>110 16 elements (64 bytes)<br>111 8 elements (32 bytes) |

Table 21-9. USBCMD Register Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 1 | RST | Controller reset. Software uses this bit to reset the controller. This bit is cleared by the controller when the reset process is complete. Software cannot terminate the reset process early by writing a zero to this register.<br>Host mode:<br>• When software sets this bit, the host controller resets its internal pipelines, timers, counters, state machines etc. to their initial value. Any transaction currently in progress on USB is immediately terminated. A USB reset is not driven on downstream ports. Software should not set this bit when USBSTS[HCH] is a zero. Attempting to reset an actively running host controller will result in undefined behavior.<br>Device mode:<br>• When software sets this bit, the USB controller resets its internal pipelines, timers, counters, state machines etc. to their initial value. Any transaction currently in progress on USB is immediately terminated. Writing a one to this bit in device mode is not recommended. |
| 0 | RS | Run/Stop.<br>Host mode:<br>• When this bit is set, the controller proceeds with the execution of the schedule. The controller continues execution as long as this bit is set. When this bit is set to 0, the host controller completes the current transaction on the USB and then halts. The USBSTS[HCH] bit indicates when the USB controller has finished the transaction and has entered the stopped state. Software should not write a one to this field unless the controller is in the halted state (that is, USBSTS[HCH] is a one).<br>Device mode:<br>• Setting this bit will cause the USB controller to enable a pull-up on D+ and initiate an attach event. This control bit is not directly connected to the pull-up enable, as the pull-up will become disabled upon transitioning into high-speed mode. Software should use this bit to prevent an attach event before the controller has been properly initialized. Clearing this bit will cause a detach event.<br>0 Stop<br>1 Run |

## 21.3.2.2 USB Status Register (USBSTS)

This register indicates various states of the USB module and any pending interrupts. This register does not indicate status resulting from a transaction on the serial bus. Software clears certain bits in this register by writing a 1 to them (indicated by a w1c in the bit's W cell in Figure 21-9).

Offset 0x144 Access: Mixed

| | 31–16 |
|---|---|
| R | — |
| W | — |
| Reset | All zeros |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | AS | PS | RCL | HCH | — | ULPII | — | SLI | SRI | URI | AAI | SEI | FRI | PCI | UEI | UI |
| W | | | | | | | | w1c | w1c | w1c | w1c | w1c | w1c | w1c | w1c | w1c |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | *n* | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Figure 21-9. USB Status Register (USBSTS)

**Table 21-10. USBSTS Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31–16 | — | Reserved, should be cleared. |
| 15 | AS | Asynchronous schedule status. Reports the current real status of the asynchronous schedule. The USB controller is not required to immediately disable or enable the asynchronous schedule when software transitions USBCMD[ASE]. When this bit and USBCMD[ASE] have the same value, the asynchronous schedule is either enabled (1) or disabled (0). Only used in host mode.<br>0 Disabled<br>1 Enabled |
| 14 | PS | Periodic schedule status. Reports the current real status of the periodic schedule. The USB controller is not required to immediately disable or enable the periodic schedule when software transitions USBCMD[PSE]. When this bit and USBCMD[PSE] have the same value, the periodic schedule is either enabled (1) or disabled (0). Only used in host mode.<br>0 Disabled<br>1 Enabled |
| 13 | RCL | Reclamation. Used to detect an empty asynchronous schedule. Only used by the host mode.<br>0 Non-empty asynchronous schedule<br>1 Empty asynchronous schedule |
| 12 | HCH | HC halted. This bit is a zero whenever USBCMD[RS] is a one. The USB controller sets this bit to one after it has stopped executing because of USBCMD[RS] being cleared, either by software or by the host controller hardware (for example, internal error). Only used in host mode.<br>0 Running<br>1 Halted |
| 11 | — | Reserved, should be cleared. |
| 10 | ULPII | ULPI interrupt. An event completion to the viewport register sets this bit. If the ULPI enables the USBINTR[ULPIE] to be set, the USB interrupt (UI) will occur. |
| 9 | — | Reserved, should be cleared. |
| 8 | SLI | DCSuspend. This is a non-EHCI bit. When a device controller enters a suspend state from an active state, this bit is set. The device controller clears the bit upon exiting from a suspend state. Only used by the device controller.<br>0 Active<br>1 Suspended |
| 7 | SRI | Host mode:<br>• This is a non-EHCI status bit. In host mode, this bit will be set every 125 us, provided the PHY clock is present and running (for example, the port is NOT suspended), and can be used by the host controller driver as a time base.<br>Device mode:<br>• SOF received. When the USB controller detects a Start Of (micro) Frame, this bit will be set. When a SOF is extremely late, the USB controller will automatically set this bit to indicate that an SOF was expected. Therefore, this bit will be set roughly every 1 msec in device FS mode and every 125 msec in HS mode and will be synchronized to the actual SOF that is received. Because the controller is initialized to FS before connect, this bit will be set at an interval of 1 msec during the prelude to the connect and chirp.<br>Software writes a 1 to this bit to clear it. |
| 6 | URI | USB reset received. This is a non-EHCI bit. When the USB controller detects a USB reset and enters the default state, this bit will be set. Software can write a 1 to this bit to clear the USB reset received status bit. Only used by the device mode.<br>0 No reset received<br>1 Reset received |

Table 21-10. USBSTS Register Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 5 | AAI | Interrupt on async advance. System software can force the controller to issue an interrupt the next time the USB controller advances the asynchronous schedule by writing a one to USBCMD[IAA]. This status bit indicates the assertion of that interrupt source. Only used by the host mode.<br>0 No async advance interrupt<br>1 Async advance interrupt |
| 4 | SEI | System error. This bit is set whenever an error is detected on the system bus. If USBINTR[SEE] is set, an interrupt will be generated. The interrupt and status bits will remain asserted until cleared by writing a 1 to this bit. Additionally, when in host mode, USBCMD[RS] is cleared, effectively disabling the USB controller. For the USB controller in device mode, an interrupt is generated, but no other action is taken.<br>0 Normal operation<br>1 Error |
| 3 | FRI | Frame list rollover. The controller sets this bit to a one when the frame list index rolls over from its maximum value to zero. The exact value at which the rollover occurs depends on the frame list size. For example. If the frame list size (as programmed in USBCMD[FS]) is 1024, FRINDEX rolls over every time FRINDEX [1 3] toggles. Similarly, if the size is 512, the USB controller sets this bit to a one every time FHINDEX [12] toggles. Only used by the host mode. |
| 2 | PCI | Host mode:<br>• Port change detect. The controller sets this bit when a connect status occurs on any port, a port enable/disable change occurs, an over current change occurs, or PORTSC[FPR] is set as the result of a J-K transition on the suspended port.<br>Device mode:<br>• The USB controller sets this bit when it enters the full or high-speed operational state. When the it exits the full or high-speed operation states due to reset or suspend events, the notification mechanisms are USBSTS[URI] and USBSTS[SLI], respectively.<br>This bit is not EHCI compatible. |
| 1 | UEI<br>(USBERRINT) | USB error interrupt (USBERRINT). When completion of a USB transaction results in an error condition, this bit is set by the controller. This bit is set along with the UI, if the TD on which the error interrupt occurred also had its interrupt on complete (IOC) bit set. See Section 4.15.1 in EHCI for a complete list of host error interrupt conditions. Also see Table 21-90 in this chapter for more information on device error matrix. For the USB controller in device mode, only resume signaling is detected, all others are ignored.<br>0 No error<br>1 Error detected |
| 0 | UI<br>(USBINT) | USB interrupt (USBINT). This bit is set by the controller when the cause of an interrupt is a completion of a USB transaction where the transfer descriptor (TD) has an interrupt on complete (IOC) bit set. This bit is also set by the controller when a short packet is detected. A short packet is when the actual number of bytes received was less than the expected number of bytes. |

### 21.3.2.3 USB Interrupt Enable Register (USBINTR)

The interrupts to software are enabled with this register. An interrupt is generated when a bit is set and the corresponding interrupt is active. The USB status register (USBSTS) still shows interrupt sources even if they are disabled by the USBINTR register, allowing polling of interrupt events by the software.

Offset 0x148 Access: Read/Write

| | 31–16 |
|---|---|
| R<br>W | — |
| Reset | All zeros |

| | 15–11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | ULPIE | — | SLE | SRE | URE | AAE | SEE | FRE | PCE | UEE | UE |
| Reset | All zeros | | | | | | | | | | | |

**Figure 21-10. USB Interrupt Enable (USBINTR)**

**Table 21-11. USBINTR Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31–11 | — | Reserved, should be cleared. |
| 10 | ULPIE | ULPI interrupt enable. An event completion to the viewport register sets the USBSTS[ULPII]. If the ULPI enables ULPIE bit to be set, then the USBINT (USBSTS[UI]) will occur.<br>0 Disable<br>1 Enable |
| 9 | — | Reserved, should be cleared. |
| 8 | SLE | Sleep enable. This is a non-EHCI bit. When this bit is a one, and USBSTS[SLI] transitions, the USB controller will issue an interrupt. The interrupt is acknowledged by software writing a one to USBSTS[SLI]. Only used in device mode.<br>0 Disable<br>1 Enable |
| 7 | SRE | SOF received enable. This is a non-EHCI bit. When this bit is a one, and USBSTS[SRI] is a one, the controller will issue an interrupt. The interrupt is acknowledged by software clearing USBSTS[SRI].<br>0 Disable<br>1 Enable |
| 6 | URE | USB reset enable. This is a non-EHCI bit. When this bit is a one, USBSTS[URI] is a one, the device controller will issue an interrupt. The interrupt is acknowledged by software clearing USBSTS[URI] bit. Only used in device mode.<br>0 Disable<br>1 Enable |
| 5 | AAE | Interrupt on async advance enable. When this bit is a one, and USBSTS[AAI] is a one, the controller will issue an interrupt at the next interrupt threshold. The interrupt is acknowledged by software clearing USBSTS[AAI]. Only used in host mode.<br>0 Disable<br>1 Enable |
| 4 | SEE | System error enable. When this bit is a one, and USBSTS[SEI] is a one, the controller will issue an interrupt. The interrupt is acknowledged by software clearing USBSTS[SEI].<br>0 Disable<br>1 Enable |
| 3 | FRE | Frame list rollover enable. When this bit is a one, and USBSTS[FRI] is a one, the controller will issue an interrupt. The interrupt is acknowledged by software clearing USBSTS[FRI]. Only used by the host mode.<br>0 Disable<br>1 Enable |

**Table 21-11. USBINTR Register Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 2 | PCE | Port change detect enable. When this bit is a one, and USBSTS[PCI] is a one, the controller will issue an interrupt. The interrupt is acknowledged by software clearing USBSTS[PCI].<br>0 Disable<br>1 Enable |
| 1 | UEE | USB error interrupt enable. When this bit is a one, and USBSTS[UEI] is a one, the controller will issue an interrupt at the next interrupt threshold. The interrupt is acknowledged by software clearing USBSTS[UEI].<br>0 Disable<br>1 Enable |
| 0 | UE | USB interrupt enable. When this bit is a one, and USBSTS[UI] is a one, the controller will issue an interrupt at the next interrupt threshold. The interrupt is acknowledged by software clearing USBSTS[UI].<br>0 Disable<br>1 Enable |

### 21.3.2.4 Frame Index Register (FRINDEX)

In host mode, this register is used by the controller to index the periodic frame list. The register updates every 125 microseconds (once each microframe). Bits N–3 are used to select a particular entry in the periodic frame list during periodic schedule execution. The number of bits used for the index depends on the size of the frame list as set by system software in USBCMD[FS].

This register must be written as a DWord. Byte writes produce-undefined results. This register cannot be written unless the USB controller is in the Halted state as indicated by the USBSTS[HCH]. A write to this register while USBCMD[RS] is set produces undefined results. Writes to this register also affect the SOF value.

In device mode, this register is read-only and, the USB controller updates the FRINDEX[13–3] register from the frame number indicated by the SOF marker. Whenever a SOF is received by the USB bus, FRINDEX[13–3] is checked against the SOF marker. If FRINDEX[13–3] is different from the SOF marker, FRINDEX[13–3] is set to the SOF value and FRINDEX[2–0] is cleared (that is, SOF for 1 msec frame). If FRINDEX[13–3] is equal to the SOF value, FRINDEX[2–0] is incremented (that is, SOF for 125-µsec microframe.)

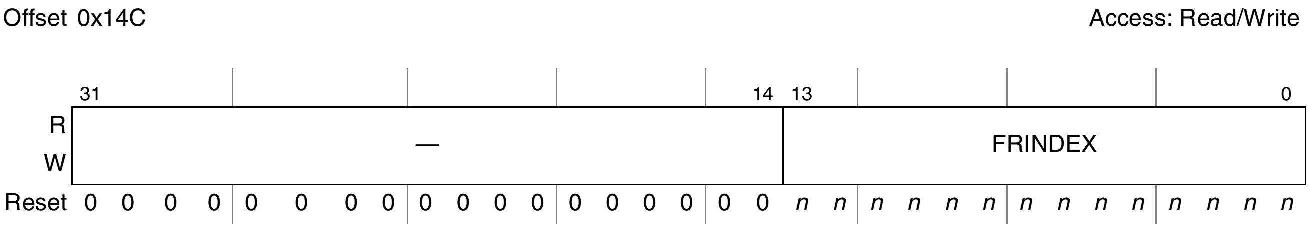

Offset 0x14C
Access: Read/Write
31
14
13
0
R
W
—
FRINDEX
Reset
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 *n n n n n n n n n n n n n n*

**Figure 21-11. USB Frame Index (FRINDEX)**

**Table 21-12. FRINDEX Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31–14 | — | Reserved, should be cleared. |
| 13–0 | FRINDEX | Frame index. The value in this register increments at the end of each time frame (for example, microframe).<br>Bits N–3 are used for the Frame List current index. This means that each location of the frame list is accessed 8 times (frames or microframes) before moving to the next index.<br>In device mode, the value is the current frame number of the last frame transmitted. It is not used as an index.<br>In either mode, bits 2–0 indicate the current microframe. |

Table 21-13 illustrates values of N based on the value of the Frame List Size in the USBCMD register, when used in host mode.

**Table 21-13. FRINDEX N Values**

| USBCMD[FS] | Frame List Size | FRINDEX N value |
|---|---|---|
| 000 | 1024 elements (4096 bytes) | 12 |
| 001 | 512 elements (2048 bytes) | 11 |
| 010 | 256 elements (1024 bytes) | 10 |
| 011 | 128 elements (512 bytes) | 9 |
| 100 | 64 elements (256 bytes) | 8 |
| 101 | 32 elements (128 bytes) | 7 |
| 110 | 16 elements (64 bytes) | 6 |
| 111 | 8 elements (32 bytes) | 5 |

### 21.3.2.5 Control Data Structure Segment Register (CTRLDSSEGMENT)

The CTRLDSSEGMENT register is not implemented on the MPC8536E.

### 21.3.2.6 Periodic Frame List Base Address Register (PERIODICLISTBASE)

This register contains the beginning address of the Periodic Frame List in the system memory. The host controller driver loads this register prior to starting the schedule execution by the controller. The memory structure referenced by this physical memory pointer is assumed to be 4-Kbyte aligned. The contents of this register are combined with the frame index register (FRINDEX) to enable the controller to step through the Periodic Frame List in sequence.

Note that this register is shared between the host and device mode functions. In host mode, it is the PERIODICLISTBASE register; in device mode, it is the DEVICEADDR register. See Section 21.3.2.7, “Device Address Register (DEVICEADDR)—Non-EHCI,” for more information.

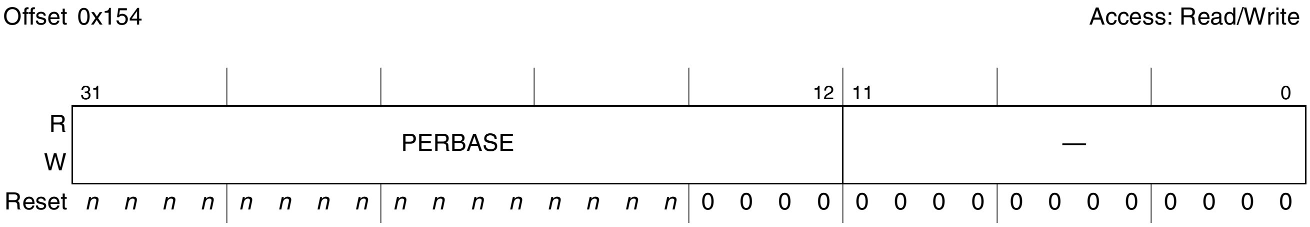

Offset 0x154
Access: Read/Write
31
12
11
0
R
W
PERBASE
—
Reset
n n n n n n n n n n n n n n n n n n n n 0 0 0 0 0 0 0 0 0 0 0 0

**Figure 21-12. Periodic Frame List Base Address (PERIODICLISTBASE)**

**Table 21-14. PERIODICLISTBASE Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31–12 | PERBASE | Base address. Correspond to memory address signal [31:12]. Only used in the host mode. |
| 11–0 | — | Reserved, should be cleared. |

### 21.3.2.7 Device Address Register (DEVICEADDR)—Non-EHCI

This register is not defined in the EHCI specification. In device mode, the upper seven bits of this register represent the device address. After any controller reset or a USB reset, the device address is set to the default address (0). The default address will match all incoming addresses. Software shall reprogram the address after receiving a SET_ADDRESS descriptor.

Note that this register is shared between the host and device mode functions. In device mode, it is the DEVICEADDR register; in host mode, it is the PERIODICLISTBASE register. See Section 21.3.2.6, “Periodic Frame List Base Address Register (PERIODICLISTBASE),” for more information.

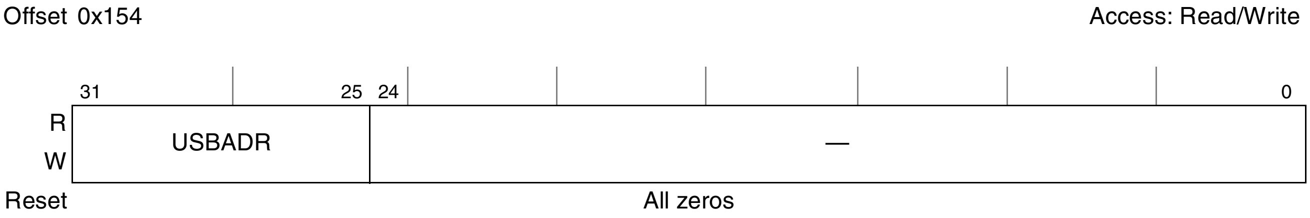


**Figure 21-13. Device Address (DEVICEADDR)**

**Table 21-15. DEVICEADDR Register Field Descriptions**

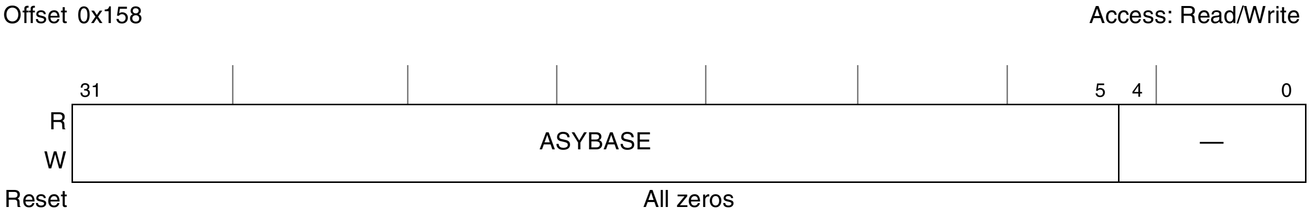
| Bits | Name | Description |
|---|---|---|
| 31–25 | USBADR | Device address. This field corresponds to the USB device address. |
| 24–0 | — | Reserved, should be cleared. |

### 21.3.2.8 Current Asynchronous List Address Register (ASYNCLISTADDR)

This 32-bit register contains the address of the next asynchronous queue head to be executed by the host. Bits 4–0 of this register cannot be modified by the system software and always return zeros when read.

Note that this register is shared between the host and device mode functions. In host mode, it is the ASYNCLISTADDR register; in device mode, it is the ENDPOINTLISTADDR register. See

Section 21.3.2.9, “Endpoint List Address Register (ENDPOINTLISTADDR)—Non-EHCI,” for more information.



Offset 0x158 Access: Read/Write

| | 31–5 | 4–0 |
|---|---|---|
| R<br>W | ASYBASE | — |
| Reset | All zeros | All zeros |

**Figure 21-14. Current Asynchronous List Address (ASYNCLISTADDR)**

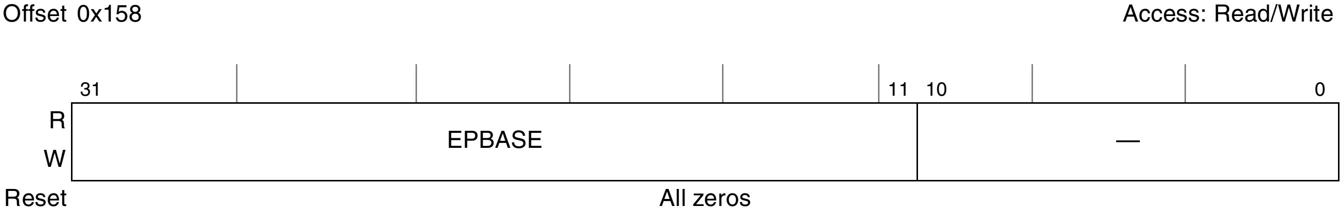
**Table 21-16. ASYNCLISTADDR Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31–5 | ASYBASE | Link pointer low (LPL). These bits correspond to memory address signals [31:5]. This field may only reference a queue head (QH). Only used by the host controller. |
| 4–0 | — | Reserved, should be cleared. |

## 21.3.2.9 Endpoint List Address Register (ENDPOINTLISTADDR)—Non-EHCI

This register is not defined in the EHCI specification. In device mode, this register contains the address of the top of the endpoint list in system memory. Bits 10–0 of this register cannot be modified by the system software and always return zeros when read. The memory structure referenced by this physical memory pointer is assumed to be 64-bytes. The queue head is actually a 48-byte structure, but must be aligned on 64-byte boundary. However, the ENDPOINTLISTADDR[EPBASE] has a granularity of 2 Kbytes, so in practice the queue head should be 2-Kbyte aligned.

Note that this register is shared between the host and device mode functions. In device mode, it is the ENDPOINTLISTADDR register; in host mode, it is the ASYNCLISTADDR register. See Section 21.3.2.8, “Current Asynchronous List Address Register (ASYNCLISTADDR),” for more information.



Offset 0x158 Access: Read/Write

| | 31–11 | 10–0 |
|---|---|---|
| R<br>W | EPBASE | — |
| Reset | All zeros | All zeros |

**Figure 21-15. Endpoint List Address (ENDPOINTLISTADDR)**

**Table 21-17. ENDPOINTLISTADDR Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31–11 | EPBASE | Endpoint list address. Address of the top of the endpoint list. |
| 10–0 | — | Reserved, should be cleared. |

### 21.3.2.10 Master Interface Data Burst Size Register (BURSTSIZE)—Non-EHCI

This register is not defined in the EHCI specification. This register is used to control and dynamically change the burst size used during data movement on the initiator (master) interface.

Offset 0x160 Access: Read/Write

| | 31 … 16 | 15 … 8 | 7 … 0 |
|---|---|---|---|
| R<br>W | — | TXPBURST | RXPBURST |
| Reset | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 0 0 1 0 0 0 0 | 0 0 0 1 0 0 0 0 |

**Figure 21-16. Master Interface Data Burst Size (BURSTSIZE)**

**Table 21-18. BURSTSIZE Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31–16 | — | Reserved, should be cleared. |
| 15–8 | TXPBURST | Programable TX burst length. This register represents the maximum length of a burst in 32-bit words while moving data from system memory to the USB bus. Must not be set to greater that 16. |
| 7–0 | RXPBURST | Programable RX burst length. This register represents the maximum length of a burst in 32-bit words while moving data from the USB bus to system memory. Must not be set to greater than 16. |

### 21.3.2.11 Transmit FIFO Tuning Controls Register (TXFILLTUNING)—Non-EHCI

This register is not defined in the EHCI specification. This register is used to control and dynamically change the burst size used during data movement on device DMA transfers. It is only used in host mode.

The fields in this register control performance tuning associated with how the USB module posts data to the TX latency FIFO before moving the data onto the USB bus. The specific areas of performance include the how much data to post into the FIFO and an estimate for how long that operation should take in the target system.

Definitions:

$T_0$ = Standard packet overhead

$T_1$ = Time to send data payload

$T_s$ = Total Packet Flight Time (send-only) packet ($T_s = T_0 + T_1$)

$T_{ff}$ = Time to fetch packet into TX FIFO up to specified level.

$T_p$ = Total Packet Time (fetch and send) packet ($T_p = T_{ff} + T_s$)

Upon discovery of a transmit (OUT/SETUP) packet in the data structures, host controller checks to ensure $T_p$ remains before the end of the [micro]frame. If so it proceeds to pre-fill the TX FIFO. If at any time during the pre-fill operation the time remaining the [micro]frame is $< T_s$ then the packet attempt ceases and the packet is tried at a later time. Although this is not an error condition and the module eventually recovers, a mark is made in the scheduler health counter to note the occurrence of a back-off event. When a back-off event is detected, the partial packet fetched may need to be discarded from the latency buffer to make room for periodic traffic that will begin after the next SOF. Too many back-off events can waste bandwidth and power on the system bus and thus should be minimized (not necessarily eliminated). Back-offs can be minimized with use of the TSCHHEALTH ($T_{ff}$) parameter described below.

Offset 0x164 Access: Read/Write

| | 31–22 | 21–16 | 15–13 | 12–8 | 7–0 |
|---|---|---|---|---|---|
| R<br>W | — | TXFIFOTHRES | — | TXSCHHEALTH | TXSCHOH |
| Reset | 0 0 0 0 0 0 0 0 0 0 | 0 0 0 0 1 0 | 0 0 0 | 0 0 0 0 0 | 0 0 0 0 0 0 0 0 |

**Figure 21-17. Transmit FIFO Tuning Controls (TXFILLTUNING)**

**Table 21-19. TXFILLTUNING Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31–22 | — | Reserved, should be cleared. |
| 21–16 | TXFIFOTHRES | FIFO burst threshold. Control the number of data bursts that are posted to the TX latency FIFO in host mode before the packet begins on to the bus. The minimum value is 2 and this value should be a low as possible to maximize USB performance. A higher value can be used in systems with unpredictable latency and/or insufficient bandwidth where the FIFO may underrun because the data transferred from the latency FIFO to USB occurs before it can be replenished from system memory. This value is ignored if USBMODE[SDIS] (stream disable bit) is set. When USBMODE[SDIS] is set, the host controller behaves as if TXFIFOTHRES is set to the maximum value. |
| 15–13 | — | Reserved, should be cleared. |
| 12–8 | TXSCHHEALTH | Scheduler health counter. Increment when the host controller fails to fill the TX latency FIFO to the level programmed by TXFIFOTHRES before running out of time to send the packet before the next Start-Of-Frame.<br>This health counter measures the number of times this occurs to provide feedback to selecting a proper TXSCHOH. Writing to this register clears the counter and this counter stops counting after reaching the maximum of 31. |
| 7–0 | TXSCHOH | Scheduler overhead. These bits add an additional fixed offset to the schedule time estimator described above as $T_{ff}$. As an approximation, the value chosen for this register should limit the number of back-off events captured in the TXSCHHEALTH to less than 10 per second in a highly utilized bus. Choosing a value that is too high for this register is not desired as it can needlessly reduce USB utilization.<br>The time unit represented in this register is 1.267µs when a device is connected in high-speed mode.<br>The time unit represented in this register is 6.333µs when a device is connected in low-/full-speed mode.<br>For most applications, TXSCHOH can be set to 4 or less. A good value to begin with is: TXFIFOTHRES × (BURSTSIZE × 4 bytes-per-word) ÷ (40 × *TimeUnit*), always rounded to the next higher integer. *TimeUnit* is either 1.267 or 6.333 as noted earlier in this description. For example, if TXFIFOTHRES is 5 and BURSTSIZE is 8, then set TXSCHOH to 5×(8×4)÷(40×1.267)=4 for a high-speed link. If this value of TXSCHOH results in a TXSCHHEALTH count of 0 per second, try lowering the value by 1 if optimizing performance is desired. If TXSCHHEALTH exceeds 10 per second, try raising the value by 1.<br>If streaming mode is disabled via the USBMODE register, treat TXFIFOTHRES as the maximum value for purposes of the TXSCHOH calculation. |

### 21.3.2.12 ULPI Register Access (ULPI VIEWPORT)

The register provides indirect access to the ULPI PHY register set. Although the controller modules perform access to the ULPI PHY register set, there may be extraordinary circumstances where software may need direct access. Be advised that writes to the ULPI through the ULPI viewport can substantially harm standard USB operations. Currently no usage model has been defined where software should need to execute writes directly to the ULPI. Note that executing read operations though the ULPI viewport should have no harmful side effects to standard USB operations. Also note that if the ULPI interface is not enabled, this register will always read zeros.

ULPI VIEWPORT is shown in Figure 21-18.

Offset 0x170 Access: Mixed

| | 31 | 30 | 29 | 28 | 27 | 26 | | 24 | 23 | | | | | | | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | ULPIWU | ULPIRUN | ULPIRW | — | ULPISS | ULPIPORT | | | ULPIADDR | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | *n* | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 15 | | | | | | | 8 | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | ULPIDATRD | | | | | | | | ULPIDTWR | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | All zeros | | | | | | | | | | | | | | | |

**Figure 21-18. ULPI Register Access (ULPI VIEWPORT)**

**Table 21-20. ULPI VIEWPORT Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31 | ULPIWU | ULPI Wake Up. Writing 1 to this bit begins the wakeup operation. This bit automatically transitions to 0 after the wakeup is complete. Once this bit is set, it can not be cleared by software.<br>**Note:** The driver must never execute a wakeup and a read/write operation at the same time. |
| 30 | ULPIRUN | ULPI Run. Writing 1 to this bit begins a read/write operation. This bit automatically transitions to 0 after the read/write is complete. Once this bit is set, it can not be cleared by software.<br>**Note:** The driver must never execute a wakeup and a read/write operation at the same time. |
| 29 | ULPIRW | This bit selects between running a read or write operation to the ULPI.<br>0 Read<br>1 Write |
| 28 | — | Reserved, should be cleared. |
| 27 | ULPISS | This bit represents the state of the ULPI interface. Before reading this bit, the ULPIPORT field should be set accordingly if used with the multi-port host. Otherwise, this field should always remain 0.<br>0 Any other state (that is, carkit, serial, low power).<br>1 Normal Sync State. |
| 26–24 | ULPIPORT | For wakeup or read/write operations this value selects the port number to which the ULPI PHY is attached. Valid values are 0 and 1. |
| 23–16 | ULPIADDR | When a read or write operation is commanded, the address of the operation is written to this field. |

Table 21-20. ULPI VIEWPORT Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 15–8 | ULPIDATRD | After a read operation completes, the result is placed in this field. |
| 7–0 | ULPIDTWR | When a write operation is commanded, the data to be sent is written to this field. |

There are two operations that can be performed with the ULPI viewport, wakeup and read /write operations. The wakeup operation is used to put the ULPI interface into normal operation mode and re-enable the clock if necessary. A wakeup operation is required before accessing the registers when the ULPI interface is operating in low power mode, serial mode, or carkit mode. The ULPI state can be determined by reading the sync state bit (ULPISS). If this bit is set, then the ULPI interface is running in normal operation mode and can accept read/write operations. If the ULPISS is cleared, then read/write operations will not be able execute. Undefined behavior results if a read or write operation is performed when ULPISS is cleared. To execute a wakeup operation, write all 32-bits of the ULPI Viewport where ULPIPORT is constructed appropriately and the ULPIWU bit is set and the ULPIRUN bit is cleared. Poll the ULPI Viewport until ULPIWU is cleared for the operation to complete.

To execute a read or write operation, write all 32-bits of the ULPI Viewport where ULPIDATWR, ULPIADDR, ULPIPORT, ULPIRW are constructed appropriately and the ULPIRUN bit is set. Poll the ULPI Viewport until ULPIRUN is cleared for the operation to complete. For read operations, ULPIDATRD is valid once ULPIRUN is cleared.

The polling method above can be replaced with interrupts using the ULPI interrupt defined in the USBSTS and USBINTR registers. When a wakeup or read/write operation completes, the ULPI interrupt is set.

### 21.3.2.13 Configure Flag Register (CONFIGFLAG)

This EHCI register is not used in this implementation. A read from this register returns a constant of a 0x0000_0001 to indicate that all port routings default to this host controller.

Offset 0x180 Access: Read only

| | 31 … 1 | 0 |
|---|---|---|
| R | — | CF |
| W | | ▨ |
| Reset | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 1 |

Figure 21-19. Configure Flag Register (CONFIGFLAG)

Table 21-21. CONFIGFLAG Register Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 31–1 | — | Reserved. |
| 0 | CF | Configure flag. Always 1 indicating all port routings default to this host. |

### 21.3.2.14 Port Status and Control Register (PORTSC)

The port status and control (PORTSC) register is only reset when power is initially applied or in response to a controller reset. The initial conditions of a port are:

- No device connected
- Port disabled

If the port has port power control, this state remains until software applies power to the port by setting port power to one.

In device mode, the USB controller does not support power control. Port control in device mode is only used for status port reset, suspend, and current connect status. It is also used to initiate test mode or force signaling and allows software to put the PHY into low power suspend mode and disable the PHY clock.

Offset 0x184 Access: Mixed

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | | | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | PTS | | — | — | PSPD | | — | PFSC | PHCD | WKOC | WKDS | WLCN | PTC | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | PIC | | PO | PP | LS | | — | PR | SUSP | FPR | OCC | OCA | PEC | PE | CSC | CCS |
| W | | | | | | | | | | | w1c | | w1c | | w1c | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | *n* | 0 | 0 |

**Figure 21-20. Port Status and Control (PORTSC)**

**Table 21-22. PORTSC Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31–30 | PTS | Port transceiver select. This register bit is used to control which parallel transceiver interface is selected.<br>00 Reserved<br>01 Reserved, should be cleared<br>10 ULPI parallel interface<br>11 Reserved<br>This bit is not defined in the EHCI specification. |
| 29 | — | Reserved, should be cleared |
| 28 | — | Reserved |
| 27–26 | PSPD | Port speed. This read-only register field indicates the speed at which the port is operating.<br>This bit is not defined in the EHCI specification.<br>00 Full-speed<br>01 Low-speed<br>10 High-speed<br>11 Undefined |
| 25 | — | Reserved, should be cleared |

**Table 21-22. PORTSC Register Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 24 | PFSC | Port force full-speed connect. Used to disable the chirp sequence that allows the port to identify itself as a HS port. This is useful for testing FS configurations with a HS host, hub or device.<br>0 Allow the port to identify itself as high speed.<br>1 Force the port to only connect at full speed.<br>This bit is not defined in the EHCI specification.<br>This bit is for debugging purposes. |
| 23 | PHCD | PHY low power suspend. This bit is not defined in the EHCI specification.<br>Host mode:<br>• The PHY can be put into low power suspend—when the downstream device has been put into suspend mode or when no downstream device is connected. Low power suspend is completely under the control of software.<br>Device mode:<br>• The PHY can be put into low power suspend—when the device is not running (USBCMD[RS] = 0b) or suspend signaling is detected on the USB. Low power suspend will be cleared automatically when the resume signaling has been detected or when forcing port resume.<br>0 Normal PHY operation.<br>1 Signal the PHY to enter low power suspend mode<br>Reading this bit indicates the status of the PHY.<br>**Note:** If there is no clock connected to the USB*n*_CLK signals, PHCD must be set and the following registers should not be written: DEVICE_ADDR/PERIODICLISTBASE, PORTSC, ENDPTCTRL0, ENDPTCTRL1, ENDPTCTRL2, ENDPTCTRL3, ENDPTCTRL4, ENDPTCTRL5. |
| 22 | WKOC | Wake on over-current enable. Writing this bit to a one enables the port to be sensitive to over-current conditions as wake-up events.<br>This field is zero if Port Power (PP) is zero.<br>This bit is (host mode only) for use by an external power control circuit. |
| 21 | WKDS | Wake on disconnect enable. Writing this bit to a one enables the port to be sensitive to device disconnects as wake-up events.<br>This field is zero if Port Power (PP) is zero or in device mode.<br>This bit is (host mode only) for use by an external power control circuit. |
| 20 | WLCN | Wake on connect enable. Writing this bit to a one enables the port to be sensitive to device connects as wake-up events.<br>This field is zero if Port Power(PP) is zero or in device mode.<br>This bit is (host mode only) for use by an external power control circuit. |
| 19–16 | PTC | Port test control. Any other value than zero indicates that the port is operating in test mode.<br>0000 Not Enabled.<br>0001 J_STATE.<br>0010 K_STATE.<br>0011 SEQ_NAK.<br>0100 Packet.<br>0101 FORCE_ENABLE.<br>0110–1111 Reserved, should be cleared.<br>Refer to Chapter 7 of the USB Specification Revision 2.0 [3] for details on each test mode. |
| 15–14 | PIC | Port indicator control. Control the link indicator signals. These signals are valid for host mode only.<br>00 Off.<br>01 Amber.<br>10 Green.<br>11 Undefined.<br>Refer to the USB Specification Revision 2.0 [3] for a description on how these bits are to be used.<br>This field is output from the module on the USB port control signals for use by an external LED driving circuit. |

**Table 21-22. PORTSC Register Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 13 | PO | Port owner. Unconditionally goes to a 0 when the configured bit in the CONFIGFLAG register makes a 0 to 1 transition. This bit unconditionally goes to 1 whenever the Configured bit is zero. System software uses this field to release ownership of the port to a selected the module (in the event that the attached device is not a high-speed device). Software writes a one to this bit when the attached device is not a high-speed device. A one in this bit means that an internal companion controller owns and controls the port.<br>Port owner hand-off is not implemented in this design, therefore this bit is always 0. |
| 12 | PP | Port power. Represents the current setting of the switch (0=off, 1=on). When power is not available on a port (that is, PP equals a 0), the port is non-functional and will not report attaches, detaches, etc.<br>When an over-current condition is detected on a powered port, the PP bit in each affected port is transitioned by the host controller driver from a one to a zero (removing power from the port).<br>This feature is implemented in the host controller (PPC = 1).<br>In a device-only implementation port power control is not necessary, thus PPC and PP = 0. |
| 11–10 | LS | Line status. Reflect the current logical levels of the USB D+ (bit 11) and D– (bit 10) signal lines. The use of line status by the host controller driver is not necessary (unlike EHCI), because the connection of FS and LS is managed by hardware.<br>00 SE0<br>10 J-state<br>01 K-state<br>11 Undefined |
| 9 | — | Reserved, should be cleared |
| 8 | PR | Port reset.<br>Host mode:<br>• When software writes a one to this bit the bus-reset sequence as defined in the USB Specification Revision 2.0 is started. This bit will automatically change to zero after the reset sequence is complete. This behavior is different from EHCI where the host controller driver is required to set this bit to a zero after the reset duration is timed in the driver.<br>Device mode:<br>• This bit is a read only status bit. Device reset from the USB bus is also indicated in the USBSTS register.<br>1 Port is in reset.<br>0 Port is not in reset.<br>This field is zero if Port Power(PP) is zero. |
| 7 | SUSP | Suspend.<br>Host mode:<br>• The port enabled bit (PE) and suspend (SUSP) bit define the port states as follows:<br>0x Disable<br>10 Enable<br>11 Suspend<br>• When in suspend state, downstream propagation of data is blocked on this port, except for port reset. The blocking occurs at the end of the current transaction if a transaction was in progress when this bit was written to 1. In the suspend state, the port is sensitive to resume detection. Note that the bit status does not change until the port is suspended and that there may be a delay in suspending a port if there is a transaction currently in progress on the USB.<br>• The module unconditionally sets this bit to zero when software clears the FPR bit. A write of zero to this bit is ignored by the host controller. If host software sets this bit to a one when the port is not enabled (that is, port enabled bit is a zero) the results are undefined.<br>• This field is zero if Port Power (PP) is zero in host mode.<br>Device mode:<br>1 Port in suspend state.<br>0 Port not in suspend state. Default.<br>• In device mode this bit is a read-only status bit. |

**Table 21-22. PORTSC Register Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 6 | FPR | Force port resume. This bit is not-EHCI compatible.<br>1 Resume detected/driven on port.<br>0 No resume (K-state) detected/driven on port.<br>Host mode:<br>• Software sets this bit to one to drive resume signaling. The controller sets this bit to one if a J-to-K transition is detected while the port is in the Suspend state. When this bit transitions to a one a J-to-K transition is detected, USBSTS[PCI] (port change detect) is also set. This bit will automatically change to zero after the resume sequence is complete. This behavior is different from EHCI where the host controller driver is required to set this bit to a zero after the resume duration is timed in the driver.<br>• Note that when the controller owns the port, the resume sequence follows the defined sequence documented in the USB Specification Revision 2.0. The resume signaling (Full-speed 'K') is driven on the port as long as this bit remains a one. This bit will remain a one until the port has switched to the high-speed idle. Writing a zero has no affect because the port controller will time the resume operation clear the bit the port control state switches to HS or FS idle.<br>• This field is zero if Port Power (PP) is zero in host mode.<br>Device mode:<br>• After the device has been in Suspend State for 5 msec or more, software must set this bit to one to drive resume signaling before clearing. The USB controller will set this bit to one if a J-to-K transition is detected while the port is in the Suspend state. The bit will be cleared when the device returns to normal operation. Also, when this bit transitions to a one because a J-to-K transition detected, USBSTS[PCI] is also set. |
| 5 | OCC | Over-current change. This bit gets set when there is a change to over-current active. Software clears this bit by writing a one to this bit position.<br>Host mode:<br>• The user can provide over-current detection to the USB*n*_PWRFAULT signal for this condition.<br>Device mode:<br>• This bit must always be 0.<br>1 Over current detect.<br>0 No over current. |
| 4 | OCA | Over-current active. This bit will automatically transition from one to zero when the over current condition is removed.<br>Host mode:<br>• The user can provide over-current detection to the USB*n*_PWRFAULT signal for this condition.<br>Device mode:<br>• This bit must always be 0.<br>1 Port currently in over-current condition.<br>0 Port not in over-current condition. |
| 3 | PEC | Port enable/disable change<br>For the root hub, this bit gets set only when a port is disabled due to disconnect on the port or due to the appropriate conditions existing at the EOF2 point (See Chapter 11 of the USB Specification). Software clears this by writing a one to it.<br>Device mode:<br>• The device port is always enabled. (This bit will be zero).<br>1 Port disabled.<br>0 No change.<br>This field is zero if Port Power(PP) is zero. |

Table 21-22. PORTSC Register Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 2 | PE | Port enabled/disabled<br>Host mode:<br>• Ports can only be enabled by the controller as a part of the reset and enable. Software cannot enable a port by writing a one to this field. Ports can be disabled by either a fault condition (disconnect event or other fault condition) or by the host software. Note that the bit status does not change until the port state actually changes. There may be a delay in disabling or enabling a port due to other host and bus events.<br>• When the port is disabled, (0) downstream propagation of data is blocked except for reset.<br>• This field is zero if Port Power(PP) is zero in host mode.<br>Device mode:<br>• The device port is always enabled. (This bit will be one). |
| 1 | CSC | Connect change status<br>Host mode:<br>• This bit indicates a change has occurred in the port's Current Connect Status. the controller sets this bit for all changes to the port device connect status, even if system software has not cleared an existing connect status change. For example, the insertion status changes twice before system software has cleared the changed condition, hub hardware will be 'setting' an already-set bit (i.e., the bit will remain set). Software clears this bit by writing a one to it.<br>1 Connect Status has changed.<br>0 No change.<br>• This field is zero if Port Power(PP) is zero.<br>Device mode:<br>• This bit is undefined. |
| 0 | CCS | Current connect status<br>Host mode:<br>1 Device is present<br>0 No device present.<br>• This field is zero if Port Power(PP) is zero in host mode.<br>Device mode:<br>1 Attached<br>0 Not attached.<br>• A one indicates that the device successfully attached and is operating in either high-speed or full-speed as indicated by the High Speed Port bit in this register. A zero indicates that the device did not attach successfully or was forcibly disconnected by the software writing a zero to USBCMD[RS] (run bit). It does not state the device being disconnected or suspended. |

### 21.3.2.15 USB Mode Register (USBMODE)—Non-EHCI

This register is not defined in the EHCI specification. This register controls the operating mode of the module.

Offset 0x1A8 Access: Read/Write

| | 31–5 | 4 | 3 | 2 | 1–0 |
|---|---|---|---|---|---|
| R<br>W | — | SDIS | SLOM | — | CM |
| Reset | All zeros | | | | |

Figure 21-21. USB Mode (USBMODE)

**Table 21-23. USBMODE Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31–5 | — | Reserved, should be cleared. |
| 4 | SDIS | Stream disable<br>Host mode:<br>• Setting this bit ensures that overruns/underruns of the latency FIFO are eliminated for low bandwidth systems where the RX and TX buffers are sufficient to contain the entire packet. Enabling stream disable also has the effect of ensuring the TX latency is filled to capacity before the packet is launched onto the USB.<br>• Note that time duration to pre-fill the FIFO becomes significant when stream disable is active. See TXFILLTUNING to characterize the adjustments needed for the scheduler when using this feature.<br>• Also note that in systems with high system bus utilization, setting this bit will ensure no overruns or underruns during operation, at the expense of link utilization. For those who desire optimal link performance, SDIS can be left clear, and the rules used under the description of the TXFILLTUNING register to limit underruns/overruns.<br>1 Active.<br>0 Inactive.<br>Device mode:<br>• Setting this bit disables double priming on both RX and TX for low bandwidth systems. This mode ensures that when the RX and TX buffers are sufficient to contain an entire packet that the standard double buffering scheme is disabled to prevent overruns/underruns in bandwidth limited systems.<br>• Note that in high-speed mode, all packets received will be responded to with a NYET handshake when stream disable is active. |
| 3 | SLOM | Setup lockout mode. In device mode, this bit controls behavior of the setup lock mechanism. See Section 21.8.3.5, “Control Endpoint Operation Model.”<br>1 Setup lockouts off. DCD requires use of setup data buffer tripwire in USBCMD (SUTW).<br>0 Setup lockouts on |
| 2 | — | Reserved, should be cleared. |
| 1–0 | CM | Controller mode<br>This register can only be written once after reset. If it is necessary to switch modes, software must reset the controller by writing to USBCMD[RST] before reprogramming this register.<br>00 Idle (default for combination host/device).<br>01 Reserved<br>10 Device controller (default for device only controller).<br>11 Host controller (default for host only controller).<br>Defaults to the idle state and needs to be initialized to the desired operating mode after reset. |

## 21.3.2.16 Endpoint Setup Status Register (ENDPTSETUPSTAT)—Non-EHCI

This register is not defined in the EHCI specification. This register contains the endpoint setup status. It is only used in device mode.

Offset 0x1AC Access: Read/Write

| | 31 … 6 | 5 … 0 |
|---|---|---|
| R<br>W | — | ENDPTSETUPSTAT |
| Reset | All zeros | |

**Figure 21-22. Endpoint Setup Status (ENDPTSETUPSTAT)**

**Table 21-24. ENDPTSETUPSTAT Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31–6 | — | Reserved, should be cleared. |
| 5–0 | ENDPTSETUP STAT | Setup endpoint status. For every setup transaction that is received, a corresponding bit in this register is set. Software must clear or acknowledge the setup transfer by writing a one to a respective bit after it has read the setup data from queue head. The response to a setup packet as in the order of operations and total response time is crucial to limit bus time outs while the setup lockout mechanism is engaged.<br>This register is only used in device mode. |

## 21.3.2.17 Endpoint Initialization Register (ENDPTPRIME)—Non-EHCI

This register is not defined in the EHCI specification. This register is used to initialize endpoints. It is only used in device mode.

Offset 0x1B0 Access: Read/Write

| | 31–22 | 21–16 | 15–6 | 5–0 |
|---|---|---|---|---|
| R<br>W | — | PETB | — | PERB |
| Reset | All zeros | | | |

**Figure 21-23. Endpoint Initialization (ENDPTPRIME)**

**Table 21-25. ENDPTPRIME Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31–22 | — | Reserved, should be cleared. |
| 21–16 | PETB | Prime endpoint transmit buffer. For each endpoint a corresponding bit is used to request that a buffer prepared for a transmit operation in order to respond to a USB IN/INTERRUPT transaction. Software should write a one to the corresponding bit when posting a new transfer descriptor to an endpoint. Hardware will automatically use this bit to begin parsing for a new transfer descriptor from the queue head and prepare a transmit buffer. Hardware will clear this bit when the associated endpoint(s) is (are) successfully primed. PETB[5] (bit 21 of the register) corresponds to endpoint 5.<br>Note that these bits will be momentarily set by hardware during hardware re-priming operations when a dTD is retired, and the dQH is updated. |
| 15–6 | — | Reserved, should be cleared. |
| 5–0 | PERB | Prime endpoint receive buffer. For each endpoint, a corresponding bit is used to request a buffer prepare for a receive operation in order to respond to a USB OUT transaction. Software should write a one to the corresponding bit whenever posting a new transfer descriptor to an endpoint. Hardware will automatically use this bit to begin parsing for a new transfer descriptor from the queue head and prepare a receive buffer. Hardware will clear this bit when the associated endpoint(s) is (are) successfully primed. PERB[5] corresponds to endpoint 5.<br>Note that these bits will be momentarily set by hardware during hardware re-priming operations when a dTD is retired, and the dQH is updated. |

### 21.3.2.18 Endpoint Flush Register (ENDPTFLUSH)—Non-EHCI

This register is not defined in the EHCI specification. This register is only used in device mode.

Offset 0x1B4 Access: Read/Write

| | 31 … 22 | 21 … 16 | 15 … 6 | 5 … 0 |
|---|---|---|---|---|
| R | — | FETB | — | FERB |
| W | | | | |
| Reset | All zeros | | | |

**Figure 21-24. Endpoint Flush (ENDPTFLUSH)**

**Table 21-26. ENDPTFLUSH Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31–22 | — | Reserved, should be cleared. |
| 21–16 | FETB | Flush endpoint transmit buffer. Writing a one to a bit(s) in this register will cause the associated endpoint(s) to clear any primed buffers. If a packet is in progress for one of the associated endpoints, then that transfer will continue until completion. Hardware will clear this register after the endpoint flush operation is successful. FETB[5] (bit 21 of the register) corresponds to endpoint 5. |
| 15–6 | — | Reserved, should be cleared. |
| 5–0 | FERB | Flush endpoint receive buffer. Writing a one to a bit(s) will cause the associated endpoint(s) to clear any primed buffers. If a packet is in progress for one of the associated endpoints, then that transfer will continue until completion. Hardware will clear this register after the endpoint flush operation is successful. FERB[5] corresponds to endpoint 5. |

### 21.3.2.19 Endpoint Status Register (ENDPTSTATUS)—Non-EHCI

This register is not defined in the EHCI specification. This register is only used in device mode.

Offset 0x1B8 Access: Read only

| | 31 … 22 | 19 … 16 | 15 … 6 | 5 … 0 |
|---|---|---|---|---|
| R | — | ETBR | — | ERBR |
| W | | | | |
| Reset | All zeros | | | |

**Figure 21-25. Endpoint Status (ENDPTSTATUS)**

**Table 21-27. ENDPTSTATUS Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31–22 | — | Reserved, should be cleared |
| 21–16 | ETBR | Endpoint transmit buffer ready. One bit for each endpoint indicates status of the respective endpoint buffer. This bit is set by the hardware as a response to receiving a command from a corresponding bit in the ENDPTPRIME register. There will always be a delay between setting a bit in the ENDPTPRIME register and endpoint indicating ready. This delay time varies based upon the current USB traffic and the number of bits set in the ENDPTPRIME register. Buffer ready is cleared by USB reset, by the USB DMA system, or through the ENDPTFLUSH register. ETBR[5] (bit 21 of the register) corresponds to endpoint 5.<br>Note that these bits will be momentarily cleared by hardware during hardware endpoint re-priming operations when a dTD is retired, and the dQH is updated. |

**Table 21-27. ENDPTSTATUS Register Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 15–6 | — | Reserved, should be cleared |
| 5–0 | ERBR | Endpoint receive buffer ready. One bit for each endpoint indicates status of the respective endpoint buffer. This bit is set by the hardware as a response to receiving a command from a corresponding bit in the ENDPTPRIME register. There will always be a delay between setting a bit in the ENDPTPRIME register and endpoint indicating ready. This delay time varies based upon the current USB traffic and the number of bits set in the ENDPTPRIME register. Buffer ready is cleared by USB reset, by the USB DMA system, or through the ENDPTFLUSH register. ERBR[5] corresponds to endpoint 5.<br>Note that these bits will be momentarily cleared by hardware during hardware endpoint re-priming operations when a dTD is retired, and the dQH is updated. |

### 21.3.2.20 Endpoint Complete Register (ENDPTCOMPLETE)—Non-EHCI

This register is not defined in the EHCI specification. This register is only used in device mode.

Offset 0x1BC Access: w1c

| | 31 22 | 21 16 | 15 6 | 5 0 |
|---|---|---|---|---|
| R | — | ETCE | — | ERCE |
| W | | w1c | | w1c |
| Reset | All zeros | | | |

**Figure 21-26. Endpoint Complete (ENDPTCOMPLETE)**

**Table 21-28. ENDPTCOMPLETE Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31–22 | — | Reserved, should be cleared |
| 21–16 | ETCE | Endpoint transmit complete event. Each bit indicates a transmit event (IN/INTERRUPT) occurred and software should read the corresponding endpoint queue to determine the endpoint status. If the corresponding IOC bit is set in the Transfer Descriptor, then this bit will be set simultaneously with the USBINT. Writing a one will clear the corresponding bit in this register. ETCE[5] (bit 21 of the register) corresponds to endpoint 5. |
| 15–6 | — | Reserved, should be cleared |
| 5–0 | ERCE | Endpoint receive complete event. Each bit indicates a received event (OUT/SETUP) occurred and software should read the corresponding endpoint queue to determine the transfer status. If the corresponding IOC bit is set in the Transfer Descriptor, then this bit will be set simultaneously with the USBINT. Writing a one will clear the corresponding bit in this register. ERCE[5] corresponds to endpoint 5. |

### 21.3.2.21 Endpoint Control Register 0 (ENDPTCTRL0)—Non-EHCI

This register is not defined in the EHCI specification. Every device will implement endpoint 0 as a control endpoint.

Offset 0x1C0 Access: Mixed

| | 31 … 24 | 23 | 22 … 20 | 19 18 | 17 | 16 | 15 … 8 | 7 | 6 … 4 | 3 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | — | TXE | — | TXT | — | TXS | — | RXE | — | RXT | — | RXS |
| W | | | | | | | | | | | | |
| Reset | 0 0 0 0 0 0 0 0 | 1 | 0 0 0 | 0 0 | 0 | 0 | 0 0 0 0 0 0 0 0 | 1 | 0 0 0 | 0 0 | 0 | 0 |

**Figure 21-27. Endpoint Control 0 (ENDPTCTRL0)**

**Table 21-29. ENDPTCTRL0 Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31–24 | — | Reserved, should be cleared. |
| 23 | TXE | TX endpoint enable. Endpoint zero is always enabled.<br>0 Disable<br>1 Enable |
| 22–20 | — | Reserved, should be cleared. |
| 19–18 | TXT | TX endpoint type. Endpoint zero is always a control endpoint (00). |
| 17 | — | Reserved, should be cleared. |
| 16 | TXS | TX endpoint stall. Software can write a one to this bit to force the endpoint to return a STALL handshake to the Host. It will continue returning STALL until the bit is cleared by software or it will automatically be cleared upon receipt of a new SETUP request.<br>1 Endpoint stalled<br>0 Endpoint OK |
| 15–8 | — | Reserved, should be cleared. |
| 7 | RXE | RX endpoint enable. Endpoint zero is always enabled.<br>0 Disabled<br>1 Enabled |
| 6–4 | — | Reserved, should be cleared. |
| 3–2 | RXT | RX endpoint type. Endpoint zero is always a control endpoint (00). |
| 1 | — | Reserved, should be cleared. |
| 0 | RXS | RX endpoint stall<br>Software can write a one to this bit to force the endpoint to return a STALL handshake to the host. It will continue returning STALL until the bit is cleared by software or it will automatically be cleared upon receipt of a new SETUP request.<br>1 Endpoint stalled<br>0 Endpoint OK |

### 21.3.2.22 Endpoint Control Register *n* (ENDPTCTRL*n*)—Non-EHCI

These registers are not defined in the EHCI specification. There is an ENDPTCTRL*n* register of each endpoint in a device.

Offset 0x1C4 (ENDPTCTRL1), 0x1C8 (ENDPTCTRL2), 0x1CA (ENDPTCTRL3),
0x1D0 (ENDPTCTRL4), 0x1D4 (ENDPTCTRL5)

Access: Read/Write

| | 31–24 | 23 | 22 | 21 | 20 | 19–18 | 17 | 16 | 15–8 | 7 | 6 | 5 | 4 | 3–2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | TXE | TXR | TXI | — | TXT | TXD | TXS | — | RXE | RXR | RXI | — | RXT | RXD | RXS |
| Reset | All zeros | | | | | | | | | | | | | | | |

**Figure 21-28. Endpoint Control 1 to 5 (ENDPTCTRL*n*)**

**Table 21-30. ENDPTCTRL*x* Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31–24 | — | Reserved, should be cleared |
| 23 | TXE | TX endpoint enable<br>0 Disabled<br>1 Enabled |
| 22 | TXR | TX data toggle reset. Whenever a configuration event is received for this endpoint, software must write a one to this bit in order to synchronize the data PID's between the Host and device. |
| 21 | TXI | TX data toggle inhibit. Used only for test and should always be written as zero. Writing a one to this bit will cause this endpoint to ignore the data toggle sequence and always transmit DATA0 for a data packet.<br>0 PID sequencing enabled<br>1 PID sequencing disabled |
| 20 | — | Reserved, should be cleared |
| 19–18 | TXT | TX endpoint type<br>00 Control<br>01 Isochronous<br>10 Bulk<br>11 Interrupt<br>**Note:** When only one endpoint (RX or TX, but not both) of an endpoint pair is used, the unused endpoint should be configured as a bulk type endpoint. |
| 17 | TXD | TX endpoint data source. This bit should always be written as 0, which selects the dual port memory/DMA engine as the source. |
| 16 | TXS | TX endpoint stall. This bit will be set automatically upon receipt of a SETUP request if this endpoint is not configured as a control endpoint. It will be cleared automatically upon receipt of a SETUP request if this endpoint is configured as a control endpoint.<br>Software can write a one to this bit to force the endpoint to return a STALL handshake to the host. It will continue to returning STALL until this bit is either cleared by software or automatically cleared as above.<br>0 Endpoint OK<br>1 Endpoint stalled |
| 15–8 | — | Reserved, should be cleared |
| 7 | RXE | RX endpoint enable<br>0 Disabled<br>1 Enabled |

**Table 21-30. ENDPTCTRL*x* Register Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 6 | RXR | RX data toggle reset. Whenever a configuration event is received for this endpoint, software must write a one to this bit in order to synchronize the data PID's between the Host and device. |
| 5 | RXI | RX data toggle inhibit. This bit is only used for test and should always be written as zero. Writing a one to this bit will cause this endpoint to ignore the data toggle sequence and always accept data packets regardless of their data PID.<br>1 PID sequencing enabled<br>0 PID sequencing disabled |
| 4 | — | Reserved, should be cleared |
| 3–2 | RXT | RX endpoint type<br>00 Control<br>01 Isochronous<br>10 Bulk<br>11 Interrupt<br>**Note:** When only one endpoint (RX or TX, but not both) of an endpoint pair is used, the unused endpoint should be configured as a bulk type endpoint. |
| 1 | RXD | RX endpoint data sink. This bit should always be written as 0, which selects the dual port memory/DMA engine as the sink. |
| 0 | RXS | RX endpoint stall. This bit will be set automatically upon receipt of a SETUP request if this endpoint is not configured as a control endpoint. It will be cleared automatically upon receipt a SETUP request if this endpoint is configured as a control endpoint,<br>Software can write a one to this bit to force the endpoint to return a STALL handshake to the host. It will continue to returning STALL until this bit is either cleared by software or automatically cleared as above,<br>1 Endpoint stalled<br>0 Endpoint OK |

### 21.3.2.23 SNOOP1 and SNOOP2—Non-EHCI

Note that these registers use big-endian byte ordering and are not defined in the EHCI specification. The SNOOP1 and SNOOP2 registers provide snooping control and address range selection function. Transactions that hit a snooping window will generate cache coherent transactions on the internal system bus. When the five lower bits (SNOOP*n*[27–31]) are equal to 00000, snooping is always disabled on the system bus for all DMA transfers. When SNOOP*n*[27–31] is 01011 through 11110, the twenty upper bits (SNOOP*n*[0–19]) provide the starting base address for which transactions are snooped. These twenty bits are compared to the twenty upper bits of the address provided by the DMA block of the USB controller. When a match occurs, the five lower bits are decoded as shown below. This provides a snooping region of 4 Kbytes to 2 Gbytes within each starting base address that is programmed by the core. The SNOOP*n*[20–26] are not used.

Offset 0x400(SNOOP1), 0x404(SNOOP2) Access: Read/Write

| | 0 19 | 20 26 | 27 31 |
|---|---|---|---|
| R | Snoop Address | — | Snoop Enables |
| W | | | |
| Reset | All zeros | | |

**Figure 21-29. Snoop 1 and Snoop 2 (SNOOP*n*)**

**Table 21-31. SNOOP*n* Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–19 | Snoop address | The starting base address for which transactions are snooped. |
| 20–26 | — | Reserved, should be cleared |
| 27–31 | Snoop Enables | 0x00 Snooping disabled<br>0x0B 4-Kbyte snoop range starting at the value defined by SNOOP*n*[0–19]<br>0x0C 8-Kbyte snoop range starting at the value defined by SNOOP*n*[0–18]<br>0x0D 16-Kbyte snoop range starting at the value defined by SNOOP*n*[0–17]<br>0x0E 32-Kbyte snoop range starting at the value defined by SNOOP*n*[0–16]<br>0x0F 64-Kbyte snoop range starting at the value defined by SNOOP*n*[0–15]<br>0x10 128-Kbyte snoop range starting at the value defined by SNOOP*n*[0–14]<br>0x11 256-Kbyte snoop range starting at the value defined by SNOOP*n*[0–13]<br>0x12 512-Kbyte snoop range starting at the value defined by SNOOP*n*[0–12]<br>0x13 1-Mbyte snoop range starting at the value defined by SNOOP*n*[0–11]<br>0x14 2-Mbyte snoop range starting at the value defined by SNOOP*n*[0–10]<br>0x15 4-Mbyte snoop range starting at the value defined by SNOOP*n*[0–9]<br>0x16 8-Mbyte snoop range starting at the value defined by SNOOP*n*[0–8]<br>0x17 16-Mbyte snoop range starting at the value defined by SNOOP*n*[0–7]<br>0x18 32-Mbyte snoop range starting at the value defined by SNOOP*n*[0–6]<br>0x19 64-M byte snoop range starting at the value defined by SNOOP*n*[0–5]<br>0x1A 31-Mbyte snoop range starting at the value defined by SNOOP*n*[0–4]<br>0x1B 256-Mbyte snoop range starting at the value defined by SNOOP*n*[0–3]<br>0x1C 512-Mbyte snoop range starting at the value defined by SNOOP*n*[0–2]<br>0x1D 1-Gbyte snoop range starting at the value defined by SNOOP*n*[0–1]<br>0x1E 2-Gbyte snoop range starting at the value defined by SNOOP*n*[0] |

### 21.3.2.24 Age Count Threshold Register (AGE_CNT_THRESH)—Non-EHCI

Note that this register uses big-endian byte ordering and is not defined in the EHCI specification. The age count threshold (AGE_CNT_THRESH) register provides the aging counter threshold value used to determine the priority state of the USB controller's internal system interface.It is only enabled if PRI_CTRL[pri_en] = 1. The threshold value is in units of platform clock/2 cycles. This register should be written during system initialization or during normal system operation when the system bus interface is idle. It can be read at any time.

If the aging counter is less than the AGE_CNT_THRESH value, default (low) priority is chosen If the aging counter is greater than or equal to the AGE_CNT_THRESH value and PRI_CTL[pri_en] = 1, an elevated priority is chosen

The aging counter begins to count from zero when a bus access is requested. It increments every bus cycle until the bus transaction completes. At the completion of a bus transaction, the counter is synchronously reset to zero. If there are any outstanding bus requests, the aging counter will then begin counting immediately.

The AGE_CNT_THRESH is compared against the value of the aging counter during each clock cycle of the current transaction. If AGE_CNT_THRESH is equal to zero, an elevated priority is always chosen. If the aging counter is less than the AGE_CNT_THRESH value, default (low) priority is selected. If the aging counter is greater than or equal to the AGE_CNT_THRESH value and PRI_CTL[pri_en] = 1, an elevated priority is chosen.

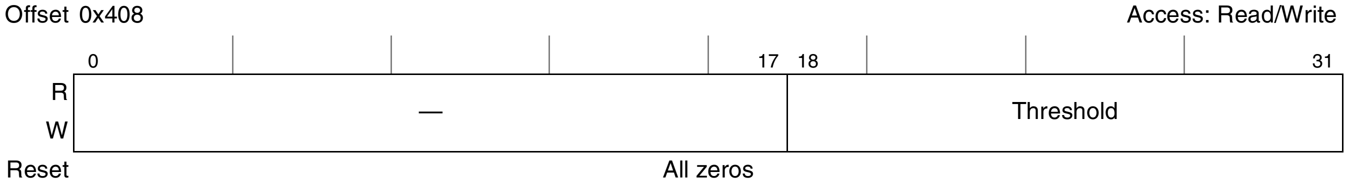


**Figure 21-30. Age Count Threshold (AGE_CNT_THRESH)**

**Table 21-32. AGE_CNT_THRESH Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–17 | — | Reserved, should be cleared |
| 18–31 | Threshold | Aging counter threshold value. |

The setting of AGE_CNT_THRESH is highly dependent on both the mix of other controllers operating on the system bus as well as the kind of traffic moving through the USB controller. A recommended approach is first to try leaving the aging mechanism disabled and see if the USB meets performance requirements. If USB performance does not meet application requirements, try the following setting:

- Set PRI_CTRL[pri_en] to 1.
- Set AGE_CNT_THRESH to 80.

Raising AGE_CNT_THRESH benefits the other controllers on the system bus by reducing the frequency that this USB controller raises its priority to the arbiter.

### 21.3.2.25 Priority Control Register (PRI_CTRL)—Non-EHCI

The priority control register (PRI_CTRL) enables dynamic priority elevation as configured in the AGE_CNT_THRESH register. Note that this register uses big-endian byte ordering and is not defined in the EHCI specification.

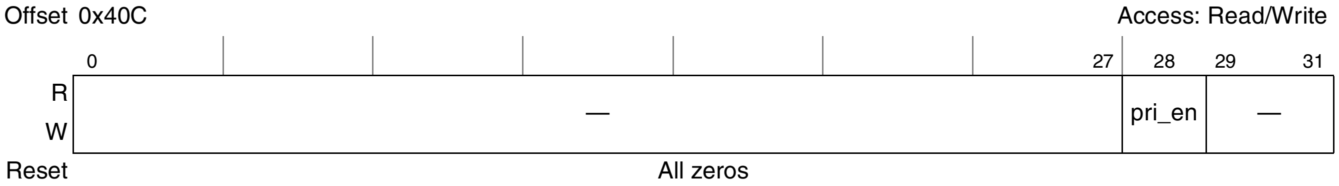


**Figure 21-31. Priority Control (PRI_CTRL)**

**Table 21-33. PRI_CTRL Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–27 | — | Reserved, should be cleared. |
| 28 | pri_en | High priority enable.<br>0 Normal operation. USB has default (low) priority.<br>1 High priority enabled. |
| 29–31 | — | Reserved. |

### 21.3.2.26 System Interface Control Register (SI_CTRL)—Non-EHCI

Note that this register uses big-endian byte ordering and is not defined in the EHCI specification. The system interface control register (SI_CTRL) controls various functions pertaining to the internal system interface.

Offset 0x410 Access: Read/Write

| | 0–26 | 27 | 28–30 | 31 |
|---|---|---|---|---|
| R<br>W | — | err_<br>disable | — | rd_prefetch<br>_val |
| Reset | All zeros | | | |

**Figure 21-32. System Interface Control Register (SI_CTRL)**

**Table 21-34. SI_CTRL Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–26 | — | Reserved, should be cleared |
| 27 | err_disable | When this bit is set, it causes the controller to ignore system bus errors. If cleared the controller responds according to the values set in USBSTS[SEI] and USBINT[SEE].<br>0 enable<br>1 disable |
| 28–30 | — | Reserved, should be cleared |
| 31 | rd_prefetch_val | Selects whether 32 bytes or 64 bytes are fetched during burst read transactions at the system interface. When this input is LOW 64 bytes are fetched and when it is HIGH 32 bytes are fetched. The setting of rd_prefetch_val must match the setting of the larger of TXPBURST and RXPBURST fields in the BURSTSIZE register. If either of these fields is 64 bytes, then rd_prefetch_val must be left cleared. Otherwise, this value should be set.<br>0 64-byte fetch.<br>1 32-byte fetch. |

### 21.3.2.27 USB General Purpose Register (CONTROL)—Non-EHCI

Note that this register uses big-endian byte ordering and is not defined in the EHCI specification. The USB general purpose (CONTROL) register contains the general-purpose IP control register outputs and is shown in Figure 21-33.

Offset 0x500 Access: Mixed

| | 0–14 | 15 |
|---|---|---|
| R | — | WU_INT |
| W | | |
| Reset | All zeros | |

| | 16–28 | 29 | 30 | 31 |
|---|---|---|---|---|
| R<br>W | — | USB_<br>EN | WU_<br>INT_EN | ULPI_<br>INT_EN |
| Reset | All zeros | | | |

**Figure 21-33. USB General-Purpose Register (CONTROL)**

Table 21-35. CONTROL Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–14 | — | Reserved |
| 15 | WU_INT | Reflects the state of the wake up interrupt. The wake up interrupt signal is asserted when a wake-up event occurs while in a low-power suspend state. If WU_INT_EN is set, this WU_INT signal generates an interrupt to the system to indicate wake up servicing is required. WU_INT will remain set until the USB controller is exited from the low power by clearing the PORTSC[PHCD] bit.<br>0 Normal operation or Low Power mode waiting for wakeup event<br>1 Low power wakeup event has occurred |
| 16–28 | — | Reserved |
| 29 | USB_EN | Used to enable the USB interface. In safe mode, all USB interface signals are put into input mode or driven inactive, except for SUSPEND_STP which is driven high. Also, the input signal USB*n*_DIR is forced to appear asserted to the controller. This prevents any start-up problems that otherwise could occur if the PHY and the controller take significantly different times to complete power-on reset.<br>1 Normal operation.<br>0 Safe mode. |
| 30 | WU_INT_EN | This bit is used to mask/unmask the system wakeup interrupt signal<br>0 System wakeup interrupt disabled<br>1 System wakeup interrupt enabled<br>**Note:** PORTSC[PHCD] bit must be set for the system wakeup interrupt generation. |
| 31 | ULPI_INT_EN | Used to enable the ULPI low power wakeup interrupt from the PHY when the PHY is in low power mode only.<br>0 ULPI low power wakeup interrupt disabled<br>1 ULPI low power wakeup interrupt enabled<br>**Note:** PORTSC[PHCD] bit must be set |

## 21.4 Functional Description

The USB DR module can be broken down into functional sub-blocks, which are described below.

### 21.4.1 System Interface

The system interface block contains all the control and status registers that allow a processor to interface to the USB module. These registers allow the processor to control the configuration of the module, ascertain the capabilities of the module, and control the module's operation. It also has registers to control snoopability and priority of the DMA interface.

### 21.4.2 DMA Engine

The module contains a local DMA engine. The DMA engine interfaces internally to the system bus. It is responsible for moving all of the data to be transferred over the USB between the module and buffers in system memory. Like the system interface block, the DMA engine block uses a simple synchronous bus signaling protocol that eases connections to a number of different standard buses.

The DMA controller must access both control information and packet data from system memory. The control information is contained in link list–based queue structures. The DMA controller has state machines that are able to parse data structures defined in the EHCI specification. In host mode, the data structures are EHCI compliant and represent queues of transfers to be performed by the host controller, including the split-transaction requests that allow an EHCI controller to direct packets to FS and LS devices. In device mode, the data structures are designed to be similar to those in the EHCI specification and are used to allow device responses to be queued for each of the active pipes in the device.

### 21.4.3 FIFO RAM Controller

The FIFO RAM controller is used for context information and to control FIFOs between the protocol engine and the DMA controller. These FIFOs decouple the system processor/memory bus requests from the extremely tight timing required by USB.

The use of the FIFO buffers differs between host and device mode operation. In host mode, a single data channel is maintained in each direction through the buffer memory. In device mode, multiple FIFO channels are maintained for each of the active endpoints in the system.

In host mode, the USB DR module uses a 512-byte Tx buffer and a 512-byte Rx buffer. Device operation uses a single 512-byte Rx buffer and a 512-byte Tx buffer for each endpoint. The 512-byte buffers allow the module to buffer a complete HS bulk packet.

### 21.4.4 PHY Interface

The USB module interfaces to any ULPI-compatible PHY. The primary function of the port controller block is to isolate the rest of the module from the transceiver, and to move all of the transceiver signaling into the primary clock domain of the module. This allows the module to run synchronously with the system processor and its associated resources.

Due to pincount limitations the module only supports certain combinations of PHY interfaces and USB functionality. Refer to Table 21-36 for more information.

**Table 21-36. Supported PHY Interfaces**

| PHY | Function |
|---|---|
| ULPI | Host/Device |

## 21.5 Host Data Structures

This section defines the interface data structures used to communicate control, status, and data between HCD (software) and the Enhanced Host Controller (hardware). The data structure definitions in this section support a 32-bit memory buffer address space. The interface consists of a periodic schedule, periodic frame list, asynchronous schedule, isochronous transaction descriptors, split-transaction isochronous transfer descriptors, queue heads, and queue element transfer descriptors.

The periodic frame list is the root of all periodic (isochronous and interrupt transfer type) support for the host controller interface. The asynchronous list is the root for all the bulk and control transfer type support. Isochronous data streams are managed using isochronous transaction descriptors. Isochronous split-transaction data streams are managed with split-transaction isochronous transfer descriptors. All interrupt, control, and bulk data streams are managed with queue heads and queue element transfer descriptors. These data structures are optimized to reduce the total memory footprint of the schedule and to reduce (on average) the number of memory accesses needed to execute a USB transaction.

Note that software must ensure that no interface data structure reachable by the EHCI host controller spans a 4K-page boundary.

The data structures defined in this section are (from the host controller's perspective) a mix of read-only and read/writable fields. The host controller must preserve the read-only fields on all data structure writes.

### 21.5.1 Periodic Frame List

Figure 21-34 shows the organization of the periodic schedule. This schedule is for all periodic transfers (isochronous and interrupt). The periodic schedule is referenced from the operational registers space using the PERIODICLISTBASE address register and the FRINDEX register. The periodic schedule is based on an array of pointers called the periodic frame list. The PERIODICLISTBASE address register is combined with the FRINDEX register to produce a memory pointer into the frame list. The periodic frame list implements a sliding window of work over time.

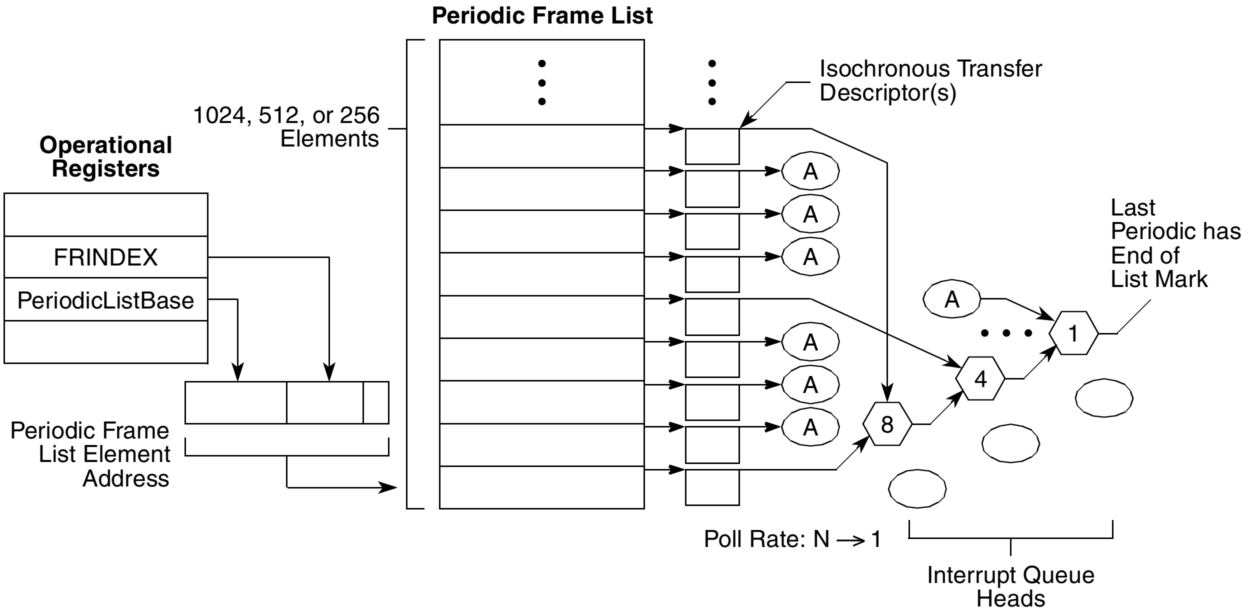

Periodic Frame List
Isochronous Transfer Descriptor(s)
1024, 512, or 256 Elements
Operational Registers
FRINDEX
PeriodicListBase
Last Periodic has End of List Mark
A
4
1
8
Periodic Frame List Element Address
Poll Rate: N → 1
Interrupt Queue Heads

**Figure 21-34. Periodic Schedule Organization**

Split transaction interrupt, bulk and control are also managed using queue heads and queue element transfer descriptors.

The periodic frame list is a 4K-page aligned array of frame list link pointers. The length of the frame list may be programmable. The programmability of the periodic frame list is exported to system software through the HCCPARAMS register. If non-programmable, the length is 1024 elements. If programmable, the length can be selected by system software as one of 256, 512, or 1024 elements. An implementation must support all three sizes. Programming the size (that is, the number of elements) is accomplished by system software writing the appropriate value into frame list size field in the USBCMD register.

Frame list link pointers direct the host controller to the first work item in the frame's periodic schedule for the current micro-frame. The link pointers are aligned on DWord boundaries within the frame list. Figure 21-35 shows the format for the frame list link pointer.

| 31 5 | 4 3 | 2 1 | 0 |
|---|---|---|---|
| Frame List Link Pointer | 00 | Typ | T |

**Figure 21-35. Frame List Link Pointer Format**

Frame list link pointers always reference memory objects that are 32-byte aligned. The referenced object may be an isochronous transfer descriptor for high-speed devices, a split-transaction isochronous transfer descriptor (for full-speed isochronous endpoints), or a queue head (used to support high-, full- and low-speed interrupt). System software should not place non-periodic schedule items into the periodic schedule. The least-significant bits in a frame list pointer are used to key the host controller in as to the type of object the pointer is referencing.

The least-significant bit is the T bit (bit 0). When this bit is set, the host controller never uses the value of the frame list pointer as a physical memory pointer. The Typ field indicates the exact type of data structure being referenced by this pointer. The value encodings for the Typ field are given in Table 21-37.

**Table 21-37. Typ Field Encodings**

| Typ | Description |
|---|---|
| 00 | Isochronous transfer descriptor |
| 01 | Queue head |
| 10 | Split transaction isochronous transfer descriptor |
| 11 | Frame span traversal node |

## 21.5.2 Asynchronous List Queue Head Pointer

The asynchronous transfer list (based at the ASYNCLISTADDR register) is where all the control and bulk transfers are managed. Host controllers use this list only when it reaches the end of the periodic list, the periodic list is disabled, or the periodic list is empty.

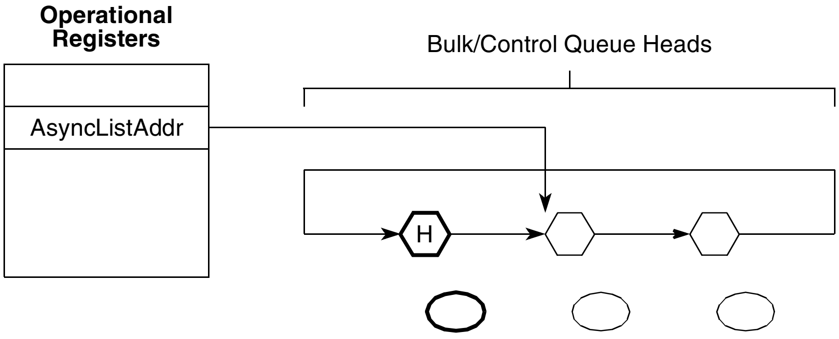


**Figure 21-36. Asynchronous Schedule Organization**

The asynchronous list is a simple circular list of queue heads. The ASYNCLISTADDR register is simply a pointer to the next queue head. This implements a pure round-robin service for all queue heads linked into the asynchronous list.

## 21.5.3 Isochronous (High-Speed) Transfer Descriptor (iTD)

Figure 21-37 illustrates the format of an isochronous transfer descriptor. This structure is used only for high-speed isochronous endpoints. All other transfer types should use queue structures. Isochronous TDs must be aligned on a 32-byte boundary.

<table>
<tr><th>31</th><th>30</th><th>29</th><th>28</th><th>27</th><th>26</th><th>25</th><th>24</th><th>23</th><th>22</th><th>21</th><th>20</th><th>19</th><th>18</th><th>17</th><th>16</th><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th><th>Offset</th></tr>
<tr><td colspan="27">Next Link Pointer</td><td colspan="2">00</td><td colspan="2">Typ</td><td>T</td><td>0x00</td></tr>
<tr><td colspan="4">Status[1]</td><td colspan="12">Transaction 0 Length[1]</td><td>ioc</td><td colspan="3">PG[2]</td><td colspan="12">Transaction 0 Offset[2]</td><td>0x04</td></tr>
<tr><td colspan="4">Status[1]</td><td colspan="12">Transaction 1 Length[1]</td><td>ioc</td><td colspan="3">PG[2]</td><td colspan="12">Transaction 1 Offset[2]</td><td>0x08</td></tr>
<tr><td colspan="4">Status[1]</td><td colspan="12">Transaction 2 Length[1]</td><td>ioc</td><td colspan="3">PG[2]</td><td colspan="12">Transaction 2 Offset[2]</td><td>0x0C</td></tr>
<tr><td colspan="4">Status[1]</td><td colspan="12">Transaction 3 Length[1]</td><td>ioc</td><td colspan="3">PG[2]</td><td colspan="12">Transaction 3 Offset[2]</td><td>0x10</td></tr>
<tr><td colspan="4">Status[1]</td><td colspan="12">Transaction 4 Length[1]</td><td>ioc</td><td colspan="3">PG[2]</td><td colspan="12">Transaction 4 Offset[2]</td><td>0x14</td></tr>
<tr><td colspan="4">Status[1]</td><td colspan="12">Transaction 5 Length[1]</td><td>ioc</td><td colspan="3">PG[2]</td><td colspan="12">Transaction 5 Offset[2]</td><td>0x18</td></tr>
<tr><td colspan="4">Status[1]</td><td colspan="12">Transaction 6 Length[1]</td><td>ioc</td><td colspan="3">PG[2]</td><td colspan="12">Transaction 6 Offset[2]</td><td>0x1C</td></tr>
<tr><td colspan="4">Status[1]</td><td colspan="12">Transaction 7 Length[1]</td><td>ioc</td><td colspan="3">PG[2]</td><td colspan="12">Transaction 7 Offset[2]</td><td>0x20</td></tr>
<tr><td colspan="20">Buffer Pointer (Page 0)</td><td colspan="4">EndPt</td><td>R</td><td colspan="7">Device Address</td><td>0x24</td></tr>
<tr><td colspan="20">Buffer Pointer (Page 1)</td><td>I/O</td><td colspan="11">Maximum Packet Size</td><td>0x28</td></tr>
<tr><td colspan="20">Buffer Pointer (Page 2)</td><td colspan="10">Reserved</td><td colspan="2">Mult</td><td>0x2C</td></tr>
<tr><td colspan="20">Buffer Pointer (Page 3)</td><td colspan="12">Reserved</td><td>0x30</td></tr>
<tr><td colspan="20">Buffer Pointer (Page 4)</td><td colspan="12">Reserved</td><td>0x34</td></tr>
<tr><td colspan="20">Buffer Pointer (Page 5)</td><td colspan="12">Reserved</td><td>0x38</td></tr>
<tr><td colspan="20">Buffer Pointer (Page 6)</td><td colspan="12">Reserved</td><td>0x3C</td></tr>
</table>

**Figure 21-37. Isochronous Transaction Descriptor (iTD)**

[1] Host controller read/write; all others read-only.

[2] These fields may be modified by the host controller if the I/O field indicates an OUT.

### 21.5.3.1 Next Link Pointer

The first DWord of an iTD is a pointer to the next schedule data structure.

Table 21-38. Next Schedule Element Pointer

| Bits | Name | Description |
|---|---|---|
| 31–5 | Link Pointer | Correspond to memory address signals [31:5], respectively. This field points to another isochronous transaction descriptor (iTD/siTD) or queue head (QH). |
| 4–3 | — | Reserved, should be cleared. These bits are reserved and their value has no effect on operation. Software should initialize this field to zero. |
| 2–1 | Typ | Indicates to the host controller whether the item referenced is an iTD, siTD or a QH. This allows the host controller to perform the proper type of processing on the item after it is fetched. Value encodings are:<br>00 iTD (isochronous transfer descriptor)<br>01 QH (queue head)<br>10 siTD (split transaction isochronous transfer descriptor)<br>11 FSTN (frame span traversal node) |
| 0 | T | Terminate<br>1 Link Pointer field is not valid.<br>0 Link Pointer field is valid. |

### 21.5.3.2 iTD Transaction Status and Control List

DWords 1–8 constitute eight slots of transaction control and status. Each transaction description includes:

- Status results field
- Transaction length (bytes to send for OUT transactions and bytes received for IN transactions).
- Buffer offset. The PG and Transaction *n* Offset fields are used with the buffer pointer list to construct the starting buffer address for the transaction.

The host controller uses the information in each transaction description, plus the endpoint information contained in the first three DWords of the buffer page pointer list, to execute a transaction on the USB.

Table 21-39. iTD Transaction Status and Control

| Bits | Name | Description |
|---|---|---|
| 31–28 | Status | Records the status of the transaction executed by the host controller for this slot. This field is a bit vector with the following encoding:<br>31 Active. Set by software to enable the execution of an isochronous transaction by the host controller. When the transaction associated with this descriptor is completed, the host controller clears this bit indicating that a transaction for this element should not be executed when it is next encountered in the schedule.<br>30 Data buffer error. Set by the host controller during status update to indicate that the host controller is unable to keep up with the reception of incoming data (overrun) or is unable to supply data fast enough during transmission (underrun). If an overrun condition occurs, no action is necessary.<br>29 Babble detected. Set by the host controller during status update when" babble" is detected during the transaction generated by this descriptor.<br>28 Transaction error (XactErr). Set by the host controller during status update in the case where the host did not receive a valid response from the device (Time-out, CRC, Bad PID, etc.). This bit may only be set for isochronous IN transactions. |
| 27–16 | Transaction *n* Length | For an OUT, this field is the number of data bytes the host controller will send during the transaction. The host controller is not required to update this field to reflect the actual number of bytes transferred during the transfer. For an IN, the initial value of the endpoint to deliver. During the status update, the host controller writes back the field is the number of bytes the host expects the number of bytes successfully received. The value in this register is the actual byte count (for example, 0 zero length data, 1 one byte, 2 two bytes, etc.). The maximum value this field may contain is 0xC00 (3072). |
| 15 | ioc | Interrupt on complete. If this bit is set, it specifies that when this transaction completes, the host controller should issue an interrupt at the next interrupt threshold. |
| 14–12 | PG | These bits are set by software to indicate which of the buffer page pointers the offset field in this slot should be concatenated to produce the starting memory address for this transaction. The valid range of values for this field is 0 to 6. |
| 11–0 | Transaction *n* Offset | This field is a value that is an offset, expressed in bytes, from the beginning of a buffer. This field is concatenated onto the buffer page pointer indicated in the adjacent PG field to produce the starting buffer address for this transaction. |

### 21.5.3.3 iTD Buffer Page Pointer List (Plus)

DWords 9–15 of an isochronous transaction descriptor are nominally page pointers (4K aligned) to the data buffer for this transfer descriptor. This data structure requires the associated data buffer to be contiguous (relative to virtual memory), but allows the physical memory pages to be non-contiguous. Seven page pointers are provided to support the expression of eight isochronous transfers. The seven pointers allow for 3 (transactions) × 1024 (maximum packet size) × 8 (transaction records) = 24576 bytes to be moved with this data structure, regardless of the alignment offset of the first page.

Since each pointer is a 4K-aligned page pointer, the least-significant 12 bits in several of the page pointers are used for other purposes.

Table 21-40. Buffer Pointer Page 0 (Plus)

| Bits | Name | Description |
|---|---|---|
| 31–12 | Buffer Pointer (Page 0) | A 4K-aligned pointer to physical memory. Corresponds to memory address bits 31–12. |
| 11–8 | EndPt | Selects the particular endpoint number on the device serving as the data source or sink. |

Table 21-40. Buffer Pointer Page 0 (Plus) (continued)

| | | |
|---|---|---|
| 7 | — | Reserved, should be cleared. Reserved for future use and should be initialized by software to zero. |
| 6–0 | Device Address | This field selects the specific device serving as the data source or sink. |

Table 21-41. iTD Buffer Pointer Page 1 (Plus)

| Bits | Name | Description |
|---|---|---|
| 31–12 | Buffer Pointer (Page 1) | This is a 4K aligned pointer to physical memory. Corresponds to memory address bits 31–12. |
| 11 | I/O | Direction (I/O). This field encodes whether the high-speed transaction should use an IN or OUT PID.<br>0 OUT<br>1 IN |
| 10–0 | Maximum Packet Size | This directly corresponds to the maximum packet size of the associated endpoint (*wMaxPacketSize*). This field is used for high-bandwidth endpoints where more than one transaction is issued per transaction description (for example, per micro-frame). This field is used with the *Multi* field to support high-bandwidth pipes. This field is also used for all IN transfers to detect packet babble. Software should not set a value larger than 1024 (0x400). Any value larger yields undefined results. |

Table 21-42. Buffer Pointer Page 2 (Plus)

| Bits | Name | Description |
|---|---|---|
| 31–12 | Buffer Pointer (Page 2) | This is a 4K-aligned pointer to physical memory. Corresponds to memory address bits 31–12. |
| 11–2 | — | Reserved, should be cleared. This bit reserved for future use and should be cleared. |
| 1–0 | Mult | Indicates to the host controller the number of transactions that should be executed per transaction description (for example, per micro-frame).<br>00 Reserved, should be cleared. A zero in this field yields undefined results.<br>01 One transaction to be issued for this endpoint per micro-frame<br>10 Two transactions to be issued for this endpoint per micro-frame<br>11 Three transactions to be issued for this endpoint per micro-frame |

Table 21-43. Buffer Pointer Page 3–6

| Bits | Name | Description |
|---|---|---|
| 31–12 | Buffer Pointer | This is a 4K aligned pointer to physical memory. Corresponds to memory address bits 31–12. |
| 11–2 | — | Reserved, should be cleared. These bits reserved for future use and should be cleared. |

### 21.5.4 Split Transaction Isochronous Transfer Descriptor (siTD)

All full-speed isochronous transfers through the internal transaction translator are managed using the siTD data structure. This data structure satisfies the operational requirements for managing the split transaction protocol.

<table>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>Offset</td></tr>
<tr><td colspan="27">Next Link Pointer</td><td colspan="2">00</td><td colspan="2">Typ</td><td>T</td><td>0x00</td></tr>
<tr><td>I/O</td><td colspan="7">Port Number</td><td>0</td><td colspan="7">Hub Address</td><td colspan="4">0000</td><td colspan="4">EndPt</td><td>0</td><td colspan="7">Device Address</td><td>0x04</td></tr>
<tr><td colspan="16">0000_0000_0000_00000</td><td colspan="8">µFrame C-mask</td><td colspan="8">µFrame S-mask</td><td>0x08</td></tr>
<tr><td>ioc</td><td>P[1]</td><td colspan="4">0000</td><td colspan="10">Total Bytes to Transfer[1]</td><td colspan="8">µFrame C-prog-mask[1]</td><td colspan="8">Status[1]</td><td>0x0C</td></tr>
<tr><td colspan="20">Buffer Pointer (Page 0)</td><td colspan="12">Current Offset[1]</td><td>0x10</td></tr>
<tr><td colspan="20">Buffer Pointer (Page 1)</td><td colspan="7">000_0000</td><td colspan="2">TP[1]</td><td colspan="3">T-count[1]</td><td>0x14</td></tr>
<tr><td colspan="27">Back Pointer</td><td colspan="4">0000</td><td>T</td><td>0x18</td></tr>
</table>

**Figure 21-38. Split-Transaction Isochronous Transaction Descriptor (siTD)**

[1] Host controller read/write; all others read-only.

### 21.5.4.1 Next Link Pointer

DWord0 of a siTD is a pointer to the next schedule data structure.

**Table 21-44. Next Link Pointer**

| Bits | Name | Description |
|---|---|---|
| 31–5 | Next Link Pointer | This field contains the address of the next data object to be processed in the periodic list and corresponds to memory address signals [31:5], respectively. |
| 4–3 | — | Reserved, should be cleared. These bits must be written as zeros. |
| 2–1 | Typ | Indicates to the host controller whether the item referenced is an iTD/siTD or a QH. This allows the host controller to perform the proper type of processing on the item after it is fetched. Value encodings are:<br>00 iTD (isochronous transfer descriptor)<br>01 QH (queue head)<br>10 siTD (split transaction isochronous transfer descriptor<br>11 FSTN (frame span traversal node) |
| 0 | T | Terminate.<br>0 Link pointer is valid.<br>1 Link pointer field is not valid. |

### 21.5.4.2 siTD Endpoint Capabilities/Characteristics

DWords 1 and 2 specify static information about the full-speed endpoint, the addressing of the parent Companion Controller, and micro-frame scheduling control.

**Table 21-45. Endpoint and Transaction Translator Characteristics**

| Bits | Name | Description |
|---|---|---|
| 31 | I/O | Direction (I/O). This field encodes whether the full-speed transaction should be an IN or OUT.<br>0 OUT<br>1 IN |
| 30–24 | Port Number | This field is the port number of the recipient transaction translator. |
| 23 | — | Reserved, should be cleared. Bit reserved and should be cleared. |
| 22–16 | Hub Address | This field holds the device address of the companion controllers' hub. |

Table 21-45. Endpoint and Transaction Translator Characteristics (continued)

| Bits | Name | Description |
|---|---|---|
| 15–12 | — | Reserved, should be cleared. Field reserved and should be cleared. |
| 11–8 | EndPt | Endpoint Number. Selects the particular endpoint number on the device serving as the data source or sink. |
| 7 | — | Reserved, should be cleared. Bit is reserved for future use. It should be cleared. |
| 6–0 | Device Address | Selects the specific device serving as the data source or sink. |

Table 21-46. Micro-Frame Schedule Control

| Bits | Name | Description |
|---|---|---|
| 31–16 | — | Reserved, should be cleared. This field reserved for future use. It should be cleared. |
| 15–8 | µFrame C-mask | Split completion mask. This field (along with the Active and SplitX- state fields in the status byte) is used to determine during which micro-frames the host controller should execute complete-split transactions. When the criteria for using this field is met, an all-zeros value has undefined behavior. The host controller uses the value of the three low-order bits of the FRINDEX register to index into this bit field. If the FRINDEX register value indexes to a position where the µFrame C-Mask field is a one, this siTD is a candidate for transaction execution. There may be more than one bit in this mask set. |
| 7–0 | µFrame S-mask | Split start mask. This field (along with the Active and SplitX-state fields in the Status byte) is used to determine during which micro-frames the host controller should execute start-split transactions. The host controller uses the value of the three low-order bits of the FRINDEX register to index into this bit field. If the FRINDEX register value indexes to a position where the µFrame S-mask field is a one, then this siTD is a candidate for transaction execution. An all zeros value in this field, in combination with existing periodic frame list has undefined results. |

### 21.5.4.3 siTD Transfer State

DWords 3–6 manage the state of the transfer.

Table 21-47. siTD Transfer Status and Control

| Bits | Name | Description |
|---|---|---|
| 31 | ioc | Interrupt on complete<br>0 Do not interrupt when transaction is complete.<br>1 Do interrupt when transaction is complete. When the host controller determines that the split transaction has completed it will assert a hardware interrupt at the next interrupt threshold. |
| 30 | P | Page select. Indicates which data page pointer should be concatenated with the CurrentOffset field to construct a data buffer pointer<br>0 Selects Page 0 pointer<br>1 Selects Page 1 pointer<br>The host controller is not required to write this field back when the siTD is retired (Active bit transitioned from a one to a zero). |
| 29–26 | — | Reserved, should be cleared. This field reserved for future use and should be cleared. |
| 25–16 | Total Bytes to Transfer | This field is initialized by software to the total number of bytes expected in this transfer. Maximum value is 1023 (3FFh) |
| 15–8 | µFrame C-prog-mask | Split complete progress mask. This field is used by the host controller to record which split-completes have been executed. |

Table 21-47. siTD Transfer Status and Control (continued)

| Bits | Name | Description | |
|---|---|---|---|
| 7–0 | Status | This field records the status of the transaction executed by the host controller for this slot. This field is a bit vector with the following encoding: | |
| | | **Status Bits** | **Definition** |
| | | 7 | Active. Set by software to enable the execution of an isochronous split transaction by the host controller. |
| | | 6 | ERR. Set by the host controller when an ERR response is received from the companion controller. |
| | | 5 | Data buffer error. Set by the host controller during status update to indicate that the host controller is unable to keep up with the reception of incoming data (overrun) or is unable to supply data fast enough during transmission (under run). In the case of an under run, the host controller will transmit an incorrect CRC (thus invalidating the data at the endpoint). If an overrun condition occurs, no action is necessary. |
| | | 4 | Babble detected. Set by the host controller during status update when” babble” is detected during the transaction generated by this descriptor. |
| | | 3 | Transaction error (XactErr). Set by the host controller during status update in the case where the host did not receive a valid response from the device (Time-out, CRC, Bad PID, etc.). This bit will only be set for IN transactions. |
| | | 2 | Missed micro-frame. The host controller detected that a host-induced hold- off caused the host controller to miss a required complete-split transaction. |
| | | 1 | Split transaction state (SplitXstate). The bit encodings are:<br>0 Do start split. This value directs the host controller to issue a Start split transaction to the endpoint when a match is encountered in the S-mask.<br>1 Do complete split. This value directs the host controller to issue a Complete split transaction to the endpoint when a match is encountered in the C-mask. |
| | | 0 | Reserved, should be cleared. Bit reserved for future use and should be cleared. |

## 21.5.4.4 siTD Buffer Pointer List (Plus)

DWords 4 and 5 are the data buffer page pointers for the transfer. This structure supports one physical page cross. The most-significant 20 bits of each DWord in this section are the 4K (page) aligned buffer pointers. The least-significant 12 bits of each DWord are used as additional transfer state.

Table 21-48. siTD Buffer Pointer Page 0 (Plus)

| Bits | Name | Description |
|---|---|---|
| 31–12 | Buffer Pointer (Page 0) | Bits 31–12 are 4K page-aligned, physical memory addresses. These bits correspond to physical address bits 31–12 respectively. The field P specifies the current active pointer |
| 11–0 | Current Offset | The 12 least-significant bits of the Page 0 pointer is the current byte offset for the current page pointer (as selected with the page indicator bit (P field)). The host controller is not required to write this field back when the siTD is retired (Active bit transitioned from a one to a zero). |

Table 21-49. siTD Buffer Pointer Page 1 (Plus)

| Bits | Name | Description |
|---|---|---|
| 31–12 | Buffer Pointer (Page 1) | Bits 31–12 are 4K page-aligned, physical memory addresses. These bits correspond to physical address bits 31–12 respectively. The field P specifies the current active pointer |
| 11–5 | — | Reserved, should be cleared. |
| 4–3 | TP | Transaction position. This field is used with T-count to determine whether to send all, first, middle, or last with each outbound transaction payload. System software must initialize this field with the appropriate starting value. The host controller must correctly manage this state during the lifetime of the transfer. The bit encodings are:<br>00 All. The entire full-speed transaction data payload is in this transaction (that is, less than or equal to 188 bytes).<br>01 Begin. This is the first data payload for a full-speed transaction that is greater than 188 bytes.<br>10 Mid. This is the middle payload for a full-speed OUT transaction that is larger than 188 bytes.<br>11 End. This is the last payload for a full-speed OUT transaction that was larger than 188 bytes. |
| 2–0 | T-Count | Transaction count. Software initializes this field with the number of OUT start-splits this transfer requires. Any value larger than 6 is undefined. |

### 21.5.4.5 siTD Back Link Pointer

DWord 6 of a siTD is simply another schedule link pointer. This pointer is always zero, or references a siTD. This pointer cannot reference any other schedule data structure.

Table 21-50. siTD Back Link Pointer

| Bits | Name | Description |
|---|---|---|
| 31–5 | Back Pointer | A physical memory pointer to an siTD |
| 4–1 | — | Reserved, should be cleared. This field is reserved for future use. It should be cleared. |
| 0 | T | Terminate<br>0 siTD Back Pointer field is valid<br>1 siTD Back Pointer field is not valid |

## 21.5.5 Queue Element Transfer Descriptor (qTD)

This data structure is only used with a queue head. This data structure is used for one or more USB transactions. This data structure is used to transfer up to 20480 (5 × 4096) bytes. The structure contains two structure pointers used for queue advancement, a DWord of transfer state, and a five-element array of data buffer pointers. This structure is 32 bytes (or one 32-byte cache line). This data structure must be physically contiguous.

The buffer associated with this transfer must be virtually contiguous. The buffer may start on any byte boundary. A separate buffer pointer list element must be used for each physical page in the buffer, regardless of whether the buffer is physically contiguous.

Host controller updates (host controller writes) to stand-alone qTDs only occur during transfer retirement. References in the following bit field definitions of updates to the qTD are to the qTD portion of a queue head.

<table>
<tr><th>31</th><th>30</th><th>29</th><th>28</th><th>27</th><th>26</th><th>25</th><th>24</th><th>23</th><th>22</th><th>21</th><th>20</th><th>19</th><th>18</th><th>17</th><th>16</th><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th><th>Offset</th></tr>
<tr><td colspan="27">Next qTD Pointer</td><td colspan="4">0000</td><td>T</td><td>0x00</td></tr>
<tr><td colspan="27">Alternate Next qTD Pointer</td><td colspan="4">0000</td><td>T</td><td>0x04</td></tr>
<tr><td>dt[1]</td><td colspan="15">Total Bytes to Transfer[1]</td><td>ioc</td><td colspan="3">C_Page[1]</td><td colspan="2">Cerr[1]</td><td colspan="2">PID Code</td><td colspan="8">Status[1]</td><td>0x08</td></tr>
<tr><td colspan="20">Buffer Pointer (Page 0)</td><td colspan="12">Current Offset[1]</td><td>0x0C</td></tr>
<tr><td colspan="20">Buffer Pointer (Page 1)</td><td colspan="12">0000_0000_0000</td><td>0x10</td></tr>
<tr><td colspan="20">Buffer Pointer (Page 2)</td><td colspan="12">0000_0000_0000</td><td>0x14</td></tr>
<tr><td colspan="20">Buffer Pointer (Page 3)</td><td colspan="12">0000_0000_0000</td><td>0x18</td></tr>
<tr><td colspan="20">Buffer Pointer (Page 4)</td><td colspan="12">0000_0000_0000</td><td>0x1C</td></tr>
</table>

**Figure 21-39. Queue Element Transfer Descriptor (qTD)**

[1] Host controller read/write; all others read-only.

Queue element transfer descriptors must be aligned on 32-byte boundaries.

### 21.5.5.1 Next qTD Pointer

The first DWord of an element transfer descriptor is a pointer to another transfer element descriptor.

**Table 21-51. qTD Next Element Transfer Pointer (DWord 0)**

| Bits | Name | Description |
|---|---|---|
| 31–5 | Next qTD Pointer | This field contains the physical memory address of the next qTD to be processed and corresponds to memory address signals [31:5], respectively. |
| 4–1 | — | Reserved, should be cleared. These bits are reserved and their value has no effect on operation. |
| 0 | T | Terminate. Indicates to the host controller that there are no more valid entries in the queue.<br>0 Pointer is valid (points to a valid transfer element descriptor)<br>1 Pointer is invalid |

### 21.5.5.2 Alternate Next qTD Pointer

The second DWord of a queue element transfer descriptor is used to support hardware-only advance of the data stream to the next client buffer on short packet. To be more explicit the host controller will always use this pointer when the current qTD is retired due to short packet.

**Table 21-52. qTD Alternate Next Element Transfer Pointer (DWord 1)**

| Bits | Name | Description |
|---|---|---|
| 31–5 | Alternate Next qTD Pointer | This field contains the physical memory address of the next qTD to be processed in the event that the current qTD execution encounters a short packet (for an IN transaction). The field corresponds to memory address signals [31:5], respectively. |
| 4–1 | — | Reserved, should be cleared. These bits are reserved and their value has no effect on operation. |
| 0 | T | Terminate. Indicates to the host controller that there are no more valid entries in the queue.<br>0 Pointer is valid (points to a valid transfer element descriptor)<br>1 Pointer is invalid |

#### 21.5.5.3 qTD Token

The third DWord of a queue element transfer descriptor contains most of the information the host controller requires to execute a USB transaction (the remaining endpoint-addressing information is specified in the queue head). Note that some of the field descriptions in Table 21-53 reference fields are defined in the queue head. See Section 21.5.6, “Queue Head,” for more information on these fields.

**Table 21-53. qTD Token (DWord 2)**

| Bits | Name | Description |
|---|---|---|
| 31 | dt | Data toggle. This is the data toggle sequence bit. The use of this bit depends on the setting of the Data Toggle Control bit in the queue head. |
| 30–16 | Total Bytes to Transfer | Total bytes to transfer. This field specifies the total number of bytes to be moved with this transfer descriptor. This field is decremented by the number of bytes actually moved during the transaction, only on the successful completion of the transaction. The maximum value software may store in this field is 5 × 4K (0x5000). This is the maximum number of bytes 5 page pointers can access. If the value of this field is zero when the host controller fetches this transfer descriptor (and the active bit is set), the host controller executes a zero-length transaction and retires the transfer descriptor. It is not a requirement for OUT transfers that total bytes to transfer be an even multiple of QH[Maximum Packet Length]. If software builds such a transfer descriptor for an OUT transfer, the last transaction will always be less than QH[Maximum Packet Length]. Although it is possible to create a transfer up to 20K this assumes the page is 0. When the offset cannot be predetermined, crossing past the 5th page can be guaranteed by limiting the total bytes to 16K. Therefore, the maximum recommended transfer is 16K (0x4000). |
| 15 | ioc | Interrupt on complete. If this bit is set, the host controller should issue an interrupt at the next interrupt threshold when this qTD is completed. |
| 14–12 | C_Page | Current rage. This field is used as an index into the qTD buffer pointer list. Valid values are in the range 0x0 to 0x4. The host controller is not required to write this field back when the qTD is retired. |

Table 21-53. qTD Token (DWord 2) (continued)

<table>
<tr><th>Bits</th><th>Name</th><th colspan="2">Description</th></tr>
<tr><td rowspan="7">11–10</td><td rowspan="7">Cerr</td><td colspan="2">Error counter. 2-bit down counter that keeps track of the number of consecutive errors detected while executing this qTD. If this field is programmed with a non-zero value during set-up, the host controller decrements the count and writes it back to the qTD if the transaction fails. If the counter counts from one to zero, the host controller marks the qTD inactive, sets the Halted bit to a one, and error status bit for the error that caused Cerr to decrement to zero. An interrupt will be generated if USBINTR[UEE] is set. If the host controller driver (HCD) software programs this field to zero during set-up, the host controller will not count errors for this qTD and there will be no limit on the retries of this qTD. Note that write-backs of intermediate execution state are to the queue head overlay area, not the qTD.</td></tr>
<tr><td><b>Error</b></td><td><b>Decrement Counter</b></td></tr>
<tr><td>Transaction Error</td><td>Yes</td></tr>
<tr><td>Data Buffer Error</td><td>No. Data buffer errors are host problems. They don't count against the device's retries. Note that software must not program Cerr to a value of zero when the EPS field is programmed with a value indicating a full- or low-speed device. This combination could result in undefined behavior.</td></tr>
<tr><td>Stalled</td><td>No. Detection of babble or stall automatically halts the queue head. Thus, count is not decremented</td></tr>
<tr><td>Babble Detected</td><td>No. Detection of babble or stall automatically halts the queue head. Thus, count is not decremented</td></tr>
<tr><td>No Error</td><td>No. If the EPS field indicates a HS device or the queue head is in the asynchronous schedule (and PIDCode indicates an IN or OUT) and a bus transaction completes and the host controller does not detect a transaction error, then the host controller should reset Cerr to extend the total number of errors for this transaction. For example, Cerr should be reset with maximum value (0b11) on each successful completion of a transaction. The host controller must never reset this field if the value at the start of the transaction is 0b00.</td></tr>
<tr><td>9–8</td><td>PID Code</td><td colspan="2">This field is an encoding of the token, which should be used for transactions associated with this transfer descriptor. Encodings are:<br>00 OUT Token generates token (E1H)<br>01 IN Token generates token (69H)<br>10 SETUP Token generates token (2DH) (undefined if endpoint is an Interrupt transfer type, for example. μFrame S-mask field in the queue head is non-zero.)<br>11 Reserved, should be cleared</td></tr>
</table>

Table 21-53. qTD Token (DWord 2) (continued)

<table>
<tr><th>Bits</th><th>Name</th><th colspan="2">Description</th></tr>
<tr><td rowspan="10">7–0</td><td rowspan="10">Status</td><td colspan="2">This field is used by the host controller to communicate individual command execution states back to the host controller driver (HCD) software. This field contains the status of the last transaction performed on this qTD. The bit encodings are:</td></tr>
<tr><th>Bits</th><th>Status Field Description</th></tr>
<tr><td>7</td><td>Active. Set by software to enable the execution of transactions by the host controller.</td></tr>
<tr><td>6</td><td>Halted. Set by the host controller during status updates to indicate that a serious error has occurred at the device/endpoint addressed by this qTD. This can be caused by babble, the error counter counting down to zero, or reception of the STALL handshake from the device during a transaction. Any time that a transaction results in the Halted bit being set, the Active bit is also cleared.</td></tr>
<tr><td>5</td><td>Data buffer error. Set by the host controller during status update to indicate that the host controller is unable to keep up with the reception of incoming data (overrun) or is unable to supply data fast enough during transmission (under run). If an overrun condition occurs, the host controller will force a time-out condition on the USB, invalidating the transaction at the source. If the host controller sets this bit to a one, then it remains a one for the duration of the transfer.</td></tr>
<tr><td>4</td><td>Babble detected. Set by the host controller during status update when babble is detected during the transaction. In addition to setting this bit, the host controller also sets the Halted bit to a one. Since babble is considered a fatal error for the transfer, setting the Halted bit to a one insures that no more transactions occur because of this descriptor.</td></tr>
<tr><td>3</td><td>Transaction error (XactErr). Set by the host controller during status update in the case where the host did not receive a valid response from the device (time-out, CRC, bad PID). If the host controller sets this bit to a one, then it remains a one for the duration of the transfer.</td></tr>
<tr><td>2</td><td>Missed micro-frame. This bit is ignored unless the QH[EPS] field indicates a full- or low-speed endpoint and the queue head is in the periodic list. This bit is set when the host controller detected that a host-induced hold-off caused the host controller to miss a required complete-split transaction. If the host controller sets this bit to a one, then it remains a one for the duration of the transfer.</td></tr>
<tr><td>1</td><td>Split transaction state (SplitXstate). This bit is ignored by the host controller unless the QH[EPS] field indicates a full- or low-speed endpoint. When a full- or low-speed device, the host controller uses this bit to track the state of the split- transaction. The functional requirements of the host controller for managing this state bit and the split transaction protocol depends on whether the endpoint is in the periodic or asynchronous schedule. The bit encodings are:<br>0 Do start split. This value directs the host controller to issue a start split transaction to the endpoint.<br>1 Do complete split. This value directs the host controller to issue a Complete split transaction to the endpoint.</td></tr>
<tr><td>0</td><td>Ping state (P)/ERR. If the QH[EPS] field indicates a high-speed device and the PID Code indicates an OUT endpoint, then this is the state bit for the Ping protocol. The bit encodings are:<br>0 Do OUT. This value directs the host controller to issue an OUT PID to the endpoint.<br>1 Do Ping. This value directs the host controller to issue a PING PID to the endpoint.<br>If the QH[EPS] field does not indicate a high-speed device, then this field is used as an error indicator bit. It is set by the host controller whenever a periodic split-transaction receives an ERR handshake.</td></tr>
</table>

### 21.5.5.4 qTD Buffer Page Pointer List

The last five DWords of a queue element transfer descriptor make up an array of physical memory address pointers. These pointers reference the individual pages of a data buffer.

System software initializes the Current Offset field to the starting offset into the current page, where current page is selected with the value in the C_Page field.

**Table 21-54. qTD Buffer Pointer**

| Bits | Name | Description |
|---|---|---|
| 31–12 | Buffer Pointer (page *n*) | Each element in the list is a 4K page aligned physical memory address. The lower 12 bits in each pointer are reserved (except for the first one), as each memory pointer must reference the start of a 4K page. The field C_Page specifies the current active pointer. When the transfer element descriptor is fetched, the starting buffer address is selected using C_Page (similar to an array index to select an array element). If a transaction spans a 4K buffer boundary, the host controller must detect the page-span boundary in the data stream, increment C_Page and advance to the next buffer pointer in the list, and conclude the transaction via the new buffer pointer. |
| 11–0 | Current Offset (Page 0)/<br>—<br>(Pages 1–4) | Reserved in all pointers except the first one (that is, Page 0). The host controller should ignore all reserved bits. For the page 0 current offset interpretation, this field is the byte offset into the current page (as selected by C_Page). The host controller is not required to write this field back when the qTD is retired. Software should ensure the reserved fields are initialized to zeros. |

## 21.5.6 Queue Head

Figure 21-40 shows the queue head structure.

<table>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>Offset</td></tr>
<tr><td colspan="27">Queue Head Horizontal Link Pointer</td><td colspan="2">00</td><td colspan="2">Typ</td><td>T</td><td>0x00</td></tr>
<tr><td colspan="4">RL</td><td>C</td><td colspan="11">Maximum Packet Length</td><td>H</td><td>dtc</td><td colspan="2">EPS</td><td colspan="4">EndPt</td><td>I</td><td colspan="7">Device Address</td><td>0x04[1]</td></tr>
<tr><td colspan="2">Mult</td><td colspan="7">Port Number</td><td colspan="7">Hub Addr</td><td colspan="8">µFrame C-mask</td><td colspan="8">µFrame S-mask</td><td>0x08[1]</td></tr>
<tr><td colspan="27">Current qTD Pointer[2]</td><td colspan="5">00000</td><td>0x0C</td></tr>
<tr><td colspan="27">Next qTD Pointer[2]</td><td colspan="4">0000</td><td>T[2]</td><td>0x10[3]</td></tr>
<tr><td colspan="27">Alternate Next qTD Pointer[2]</td><td colspan="4">NakCnt[2]</td><td>T[2]</td><td>0x14[3,4]</td></tr>
<tr><td>dt[1]</td><td colspan="15">Total Bytes to Transfer[2]</td><td>ioc[2]</td><td colspan="3">C_Page[2]</td><td colspan="2">Cerr[2]</td><td colspan="2">PID Code[2]</td><td colspan="8">Status[2]</td><td>0x18[3,4]</td></tr>
<tr><td colspan="20">Buffer Pointer (Page 0)[2]</td><td colspan="12">Current Offset[2]</td><td>0x1C[3,4]</td></tr>
<tr><td colspan="20">Buffer Pointer (Page 1)[2]</td><td colspan="4">0000</td><td colspan="8">C-prog-mask[2]</td><td>0x20[3,4]</td></tr>
<tr><td colspan="20">Buffer Pointer (Page 2)[2]</td><td colspan="7">S-bytes[2]</td><td colspan="5">FrameTag[2]</td><td>0x24[3,4]</td></tr>
<tr><td colspan="20">Buffer Pointer (Page 3)[2]</td><td colspan="12">0000_0000_0000</td><td>0x28[3]</td></tr>
<tr><td colspan="20">Buffer Pointer (Page 4)[2]</td><td colspan="12">0000_0000_0000</td><td>0x2C[3]</td></tr>
</table>

**Figure 21-40. Queue Head Layout**

[1] Offsets 0x04 through 0x0B contain the static endpoint state.

[2] Host controller read/write; all others read-only.

[3] Offsets 0x10 through 0x2F contain the transfer overlay.

[4] Offsets 0x14 through 0x27 contain the transfer results.

### 21.5.6.1 Queue Head Horizontal Link Pointer

The first DWord of a queue head contains a link pointer to the next data object to be processed after any required processing in this queue has been completed, as well as the control bits defined below.

This pointer may reference a queue head or one of the isochronous transfer descriptors. It must not reference a queue element transfer descriptor.

**Table 21-55. Queue Head DWord 0**

| Bits | Name | Description |
|---|---|---|
| 31–5 | QHLP | Queue head horizontal link pointer. This field contains the address of the next data object to be processed in the horizontal list and corresponds to memory address signals [31:5], respectively. |
| 4–3 | — | Reserved, should be cleared. These bits must be written as zeros. |
| 2–1 | Typ | Indicates to the hardware whether the item referenced by the link pointer is an iTD, siTD or a QH. This allows the host controller to perform the proper type of processing on the item after it is fetched.<br>00 iTD (isochronous transfer descriptor)<br>01 QH (queue head)<br>10 siTD (split transaction isochronous transfer descriptor)<br>11 FSTN (frame span traversal node) |
| 0 | T | Terminate.<br>1 Last QH (pointer is invalid).<br>0 Pointer is valid.<br>If the queue head is in the context of the periodic list, a one bit in this field indicates to the host controller that this is the end of the periodic list. This bit is ignored by the host controller when the queue head is in the asynchronous schedule. Software must ensure that queue heads reachable by the host controller always have valid horizontal link pointers. |

### 21.5.6.2 Endpoint Capabilities/Characteristics

The second and third DWords of a Queue Head specify static information about the endpoint. This information does not change over the lifetime of the endpoint. There are three types of information in this region:

- Endpoint characteristics. These are the USB endpoint characteristics, which include addressing, maximum packet size, and endpoint speed.
- Endpoint capabilities. These are adjustable parameters of the endpoint. They affect how the endpoint data stream is managed by the host controller.
- Split transaction characteristics. This data structure manages full- and low-speed data streams for bulk, control, and interrupt with split transactions to USB 2.0 Hub transaction translator. Additional fields exist for addressing the hub and scheduling the protocol transactions (for periodic).

The host controller must not modify the bits in this region.

Table 21-56. Endpoint Characteristics: Queue Head DWord 1

| Bits | Name | Description |
|---|---|---|
| 31–28 | RL | Nak count reload. This field contains a value, which is used by the host controller to reload the Nak Counter field. |
| 27 | C | Control endpoint flag. If the QH[EPS] field indicates the endpoint is not a high-speed device, and the endpoint is a control endpoint, then software must set this bit to a one. Otherwise, it should always set this bit to a zero. |
| 26–16 | Maximum Packet Length | This directly corresponds to the maximum packet size of the associated endpoint (wMaxPacketSize). The maximum value this field may contain is 0x400 (1024). |
| 15 | H | Head of reclamation list flag. This bit is set by system software to mark a queue head as being the head of the reclamation list. |
| 14 | dtc | Data toggle control (DTC). Specifies where the host controller should get the initial data toggle on an overlay transition.<br>0 Ignore DT bit from incoming qTD. Host controller preserves DT bit in the queue head.<br>1 Initial data toggle comes from incoming qTD DT bit. Host controller replaces DT bit in the queue head from the DT bit in the qTD. |
| 13–12 | EPS | Endpoint speed. This is the speed of the associated endpoint.<br>00 Full-speed (12 Mbps)<br>01 Low-speed (1.5 Mbps)<br>10 High-speed (480 Mbps)<br>11 Reserved, should be cleared This field must not be modified by the host controller. |
| 11–8 | EndPt | Endpoint number. Selects the particular endpoint number on the device serving as the data source or sink. |
| 7 | I | Inactivate on next transaction. This bit is used by system software to request that the host controller set the Active bit to zero. This field is only valid when the queue head is in the periodic schedule and the EPS field indicates a full- or low-speed endpoint. Setting this bit when the queue head is in the asynchronous schedule or the EPS field indicates a high-speed device yields undefined results. |
| 6–0 | Device Address | Selects the specific device serving as the data source or sink. |

Table 21-57. Endpoint Capabilities: Queue Head DWord 2

| Bits | Name | Description |
|---|---|---|
| 31–30 | Mult | High-bandwidth pipe multiplier. This field is a multiplier used to key the host controller as the number of successive packets the host controller may submit to the endpoint in the current execution. The host controller makes the simplifying assumption that software properly initializes this field (regardless of location of queue head in the schedules or other run time parameters).<br>00 Reserved, should be cleared. A zero in this field yields undefined results.<br>01 One transaction to be issued for this endpoint per micro-frame<br>10 Two transactions to be issued for this endpoint per micro-frame<br>11 Three transactions to be issued for this endpoint per micro-frame |
| 29–23 | Port Number | This field is ignored by the host controller unless the EPS field indicates a full- or low-speed device. The value is the port number identifier on the USB 2.0 hub (for hub at device address Hub Addr below), below which the full- or low-speed device associated with this endpoint is attached. This information is used in the split-transaction protocol. |
| 22–16 | Hub Addr | This field is ignored by the host controller unless the EPS field indicates a full-or low-speed device. The value is the USB device address of the USB 2.0 hub below which the full- or low-speed device associated with this endpoint is attached. This field is used in the split-transaction protocol. |

Table 21-57. Endpoint Capabilities: Queue Head DWord 2 (continued)

| Bits | Name | Description |
|---|---|---|
| 15–8 | µFrame C-mask | This field is ignored by the host controller unless the EPS field indicates this device is a low- or full-speed device and this queue head is in the periodic list. This field (along with the Active and SplitX-state fields) is used to determine during which micro-frames the host controller should execute a complete-split transaction. When the criteria for using this field are met, a zero value in this field has undefined behavior. This field is used by the host controller to match against the three low-order bits of the FRINDEX register. If the FRINDEX register bits decode to a position where the µFrame C- mask field is a one, then this queue head is a candidate for transaction execution. There may be more than one bit in this mask set. |
| 7–0 | µFrame S-mask | Interrupt schedule mask. This field is used for all endpoint speeds. Software should set this field to a zero when the queue head is on the asynchronous schedule. A non-zero value in this field indicates an interrupt endpoint. The host controller uses the value of the three low-order bits of the FRINDEX register as an index into a bit position in this bit vector. If the µFrame S-mask field has a one at the indexed bit position then this queue head is a candidate for transaction execution. If the EPS field indicates the endpoint is a high-speed endpoint, then the transaction executed is determined by the PID_Code field contained in the execution area. This field is also used to support split transaction types: Interrupt (IN/OUT). This condition is true when this field is non-zero and the EPS field indicates this is either a full- or low-speed device. A zero value in this field, in combination with existing in the periodic frame list has undefined results. |

### 21.5.6.3 Transfer Overlay

The nine DWords in this area represent a transaction working space for the host controller. The general operational model is that the host controller can detect whether the overlay area contains a description of an active transfer. If it does not contain an active transfer, then it follows the queue head horizontal link pointer to the next queue head. The host controller will never follow the next transfer queue element or alternate queue element pointers unless it is actively attempting to advance the queue. For the duration of the transfer, the host controller keeps the incremental status of the transfer in the overlay area. When the transfer is complete, the results are written back to the original queue element.

The DWord3 of a queue head contains a pointer to the source qTD currently associated with the overlay. The host controller uses this pointer to write back the overlay area into the source qTD after the transfer is complete.

Table 21-58. Current qTD Link Pointer

| Bits | Name | Description |
|---|---|---|
| 31–5 | Current qTD Pointer | Current element transaction descriptor link pointer. This field contains the address Of the current transaction being processed in this queue and corresponds to memory address signals [31:5], respectively. |
| 4–0 | — | Reserved, should be cleared. These bits are ignored by the host controller when using the value as an address to write data. The actual value may vary depending on the usage. |

The DWords 4–11 of a queue head are the transaction overlay area. This area has the same base structure as a queue element transfer descriptor. The queue head utilizes the reserved fields of the page pointers to implement tracking the state of split transactions.

This area is characterized as an overlay because when the queue is advanced to the next queue element, the source queue element is merged onto this area. This area serves an execution cache for the transfer.

**Table 21-59. Host-Controller Rules for Bits in Overlay (DWords 5, 6, 8, and 9)**

| DWord | QH Offset | Bits | Name | Description |
|---|---|---|---|---|
| 5 | 0x14 | 4–1 | NakCnt | Nak counter—RW. This field is a counter the host controller decrements whenever a transaction for the endpoint associated with this queue head results in a Nak or Nyet response. This counter is reloaded from RL before a transaction is executed during the first pass of the reclamation list (relative to an Asynchronous List Restart condition). It is also loaded from RL during an overlay. |
| 6 | 0x18 | 31 | dt | Data toggle. The Data toggle control controls whether the host controller preserves this bit when an overlay operation is performed. |
| 6 | 0x18 | 15 | ioc | Interrupt on complete. The ioc control bit is always inherited from the source qTD when the overlay operation is performed. |
| 6 | 0x18 | 11–10 | Cerr | Error counter. Copied from the qTD during the overlay and written back during queue advancement. |
| 6 | 0x18 | 0 | Status[0] | Ping state (P)/ERR. If the EPS field indicates a high-speed endpoint, then this field should be preserved during the overlay operation. |
| 8 | 0x20 | 7–0 | C-prog-mask | Split-transaction complete-split progress. Initialized to zero during any overlay. This field is used to track the progress of an interrupt split-transaction. |
| 9 | 0x24 | 11–5 | S-bytes | Software must ensure that the S-bytes field in a qTD is zero before activating the qTD. Keeps track of the number of bytes sent or received during an IN or OUT split transaction. |
| 9 | 0x24 | 4–0 | FrameTag | Split-transaction frame tag. Initialized to zero during any overlay. This field is used to track the progress of an interrupt split-transaction. |

## 21.5.7 Periodic Frame Span Traversal Node (FSTN)

The periodic frame span traversal node (FSTN) data structure is to be used only for managing full- and low-speed transactions that span a host-frame boundary. Software must not use an FSTN in the asynchronous schedule. An FSTN in the asynchronous schedule results in undefined behavior. Software must not use the FSTN feature with a host controller whose HCIVERSION register indicates a revision implementation under 0x0096. Note that FSTNs were not defined for EHCI implementations before Revision 0.96 of the EHCI Specification and their use may yield undefined results.

| 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 | 4 3 | 2 1 | 0 | Offset |
|---|---|---|---|---|
| Normal Path Link Pointer | 00 | Typ | T | 0x00 |
| Back Path Link Pointer | 00 | Typ | T | 0x04 |

**Figure 21-41. Frame Span Traversal Node Structure**

#### 21.5.7.1 FTSN Normal Path Pointer

The first DWord of an FSTN contains a link pointer to the next schedule object. This object can be of any valid periodic schedule data type.

Table 21-60. FTSN Normal Path Pointer

| Bits | Name | Description |
|---|---|---|
| 31–5 | NPLP | Normal path link pointer. Contains the address of the next data object to be processed in the periodic list and corresponds to memory address signals [31:5], respectively. |
| 4–3 | — | Reserved, should be cleared. These bits must be written as 0s. |
| 2–1 | Typ | Indicates to the host controller whether the item referenced is a iTD/siTD, QH, or FSTN. This allows the host controller to perform the proper type of processing on the item after it is fetched.<br>00 iTD (isochronous transfer descriptor)<br>01 QH (queue head)<br>10 siTD (split transaction isochronous transfer descriptor)<br>11 FSTN (frame span traversal node) |
| 0 | T | Terminate.<br>0 Link pointer is valid.<br>1 Link pointer field is not valid. |

#### 21.5.7.2 FSTN Back Path Link Pointer

The second DWord of an FTSN node contains a link pointer to a queue head. If the T-bit in this pointer is a zero, then this FSTN is a Save-Place indicator. Its Typ field must be set by software to indicate the target data structure is a queue head. If the T-bit in this pointer is set, then this FSTN is the Restore indicator. When the T-bit is a one, the host controller ignores the Typ field.

Table 21-61. FSTN Back Path Link Pointer

| Bits | Name | Description |
|---|---|---|
| 31–5 | BPLP | Back path link pointer. Contains the address of a queue head. This field corresponds to memory address signals [31:5], respectively. |
| 4–3 | — | Reserved, should be cleared. These bits must be written as 0s. |
| 2–1 | Typ | Software must ensure this field is set to indicate the target data structure is a Queue Head (01). Any other value in this field yields undefined results. |
| 0 | T | Terminate.<br>0 Link pointer is valid (that is, the host controller may use bits 31–5 (in combination with the CTRLDSSEGMENT register if applicable) as a valid memory address). This value also indicates that this FSTN is a Save-Place indicator.<br>1 Link pointer field is not valid (that is, the host controller must not use bits 31–5 (in combination with the CTRLDSSEGMENT register if applicable) as a valid memory address). This value also indicates that this FSTN is a Restore indicator. |

## 21.6 Host Operations

The general operational model for the USB module in host mode is defined by the Enhanced Host Controller Interface (EHCI) Specification. The EHCI specification describes the register-level interface for a host controller for the USB Revision 2.0. It includes a description of the hardware/software interface between system software and host controller hardware. Information concerning the initialization of the USB module is included in the following section; however, the full details of the EHCI specification are beyond the scope of this document.

### 21.6.1 Host Controller Initialization

After initial power-on or host controller reset (hardware or through USBCMD[RST]), all of the operational registers will be at their default values, as illustrated in Table 25. After a hardware reset, only the operational registers not contained in the auxiliary power well will be at their default values.

**Table 21-62. Default Values of Operational Register Space**

| Operational Register | Default Value (After Reset) |
|---|---|
| USBCMD | 0x0008_0000 (0x0008_0B00 if asynchronous schedule park capability is set) |
| USBSTS | 0x0000_1000 |
| USBINTR | 0x0000_0000 |
| FRINDEX | 0x0000_0000 |
| CTRLDSSEGMENT | 0x0000_0000 |
| PERIODICLISTBASE | Undefined |
| ASYNCLISTADDR | Undefined |
| CONFIGFLAG | 0x0000_0000 |
| PORTSC | 0x0000_2000 (w/PPC set);<br>0x0000_3000 (w/PPC cleared) |

In order to initialize the USB DR module, software should perform the following steps

1. Set the controller mode to host mode. Optionally set USBMODE[SDIS] (streaming disable)

**NOTE**

Transitioning from device mode to host mode requires a host controller reset before modifying USBMODE.

2. Optionally modify the BURSTSIZE register.
3. Program the PTS field of the PORTSC register if using a non-ULPI PHY.
4. Set CONTROL[USB_EN].
5. Write the appropriate value to the USBINTR register to enable the appropriate interrupts.
6. Write the base address of the periodic frame list to the PERIODICLIST BASE register. If there are no work items in the periodic schedule, all elements of the periodic frame list should have their T-Bits set.
7. Write the USBCMD register to set the desired interrupt threshold, frame list size (if applicable) and turn the controller by setting the RS bit.

At this point, the USB module is up and running and the port registers begin reporting device connects. System software can enumerate a port through the reset process (where the port is in the enabled state). At this point, the port is active with SOFs occurring down the enabled port enabled high-speed ports, but the schedules have not yet been enabled. The EHCI host controller will not transmit SOFs to enabled Full- or Low-speed ports.

In order to communicate with devices via the asynchronous schedule, system software must write the ASYNDLISTADDR register with the address of a control or bulk queue head. Software must then enable the asynchronous schedule by writing a one to USBCMD[ASE]. In order to communicate with devices via the periodic schedule, system software must enable the periodic schedule by writing a one to USBCMD[PSE]. Note that the schedules can be turned on before the first port is reset (and enabled).

Any time the USBCMD register is written, system software must ensure the appropriate bits are preserved, depending on the intended operation.

### 21.6.2 Power Port

The HCSPARAMS[PPC] bit indicates whether the USB 2.0 host controller has port power control. When the PPC bit is set, the host controller supports port power switches. Each available switch has an output enable. PPE is controlled based on the state of the combination bits PPC bit, EHCI Configured (CF)-bit and individual Port Power (PP) bits.

### 21.6.3 Reporting Over-Current

Host ports by definition are power providers on USB. Whether the ports are considered high- or low-powered is a platform implementation issue. The EHCI PORTSC register has an over-current status and over-current change bit. The functionality of these bits is specified in the USB Specification Revision 2.0.

The over current detection and limiting logic resides outside the USB logic. The over-current condition effects the following bits in the PORTSC register on the EHCI port:

- Over-current active bit (OCA) is set. When the over-current condition goes away, the OCA will transition from a one to a zero.
- Over-current change bit (OCC) is set. On every transition of OCA, the controller will set OCC to a one. Software sets OCC to a zero by writing a one to this bit.
- Port enabled/disabled bit (PE) is cleared. When this change bit gets set, USBSTS[PCI] (the port change detect bit) is set.
- Port power (PP) bit may optionally be cleared. There is no requirement in USB that a power provider shut off power in an over current condition. It is sufficient to limit the current and leave power applied. When OCC transitions from a zero to a one, the controller also sets USBSTS[PCI] to a one. In addition, if the Port Change Interrupt Enable bit, USBINTR[PCE], is a one, the controller issues an interrupt to the system. Refer to Table 21-63 for summary of behavior for over-current detection when the controller is halted (suspended from a device component point of view).

### 21.6.4 Suspend/Resume

The host controller provides an equivalent suspend and resume model as that defined for individual ports in a USB 2.0 hub. Control mechanisms are provided to allow system software to suspend and resume individual ports. The mechanisms allow the individual ports to be resumed completely through software initiation. Other control mechanisms are provided to parameterize the host controller's response (or sensitivity) to external resume events. In this discussion, host-initiated, or software-initiated resumes are called Resume Events/Actions; bus-initiated resume events are called wake-up events. The classes of wakeup events are:

- Remote-wakeup enabled device asserts resume signaling. In similar kind to USB 2.0 hubs, when in host mode the host controller responds to explicit device resume signaling and wake up the system (if necessary).
- Port connect and disconnect and over-current events. Sensitivity to these events can be turned on or off by using the port control bits in the PORTSC register.

Selective suspend is a feature supported by the PORTSC register. It is used to place specific ports into a suspend mode. This feature is used as a functional component for implementing the appropriate power management policy implemented in a particular operating system. When system software intends to suspend the bus, it should suspend the enabled port, then shut off the controller by setting the USBCMD[RS] to a zero.

When a wake event occurs the system will resume operation and system software must set the RS bit to a one and resume the suspended port.

### 21.6.4.1 Port Suspend/Resume

System software places the USB into suspend mode by writing a one into the appropriate PORTSC Suspend bit. Software must only set the Suspend bit when the port is in the enabled state (Port Enabled bit is a one).

The host controller may evaluate the Suspend bit immediately or wait until a micro-frame or frame boundary occurs. If evaluated immediately, the port is not suspended until the current transaction (if one is executing) completes. Therefore, there may be several micro-frames of activity on the port until the host controller evaluates the Suspend bit. The host controller must evaluate the Suspend bit at least every frame boundary.

System software can initiate a resume on the suspended port by writing a one to PORTSC[FPR]. Software should not attempt to resume a port unless the port reports that it is in the suspended state. If system software sets PORTSC[FPR] when the port is not in the suspended state, the resulting behavior is undefined. In order to assure proper USB device operation, software must wait for at least 10 milliseconds after a port indicates that it is suspended (Suspend bit is a one) before initiating a port resume through PORTSC[FPR]. When PORTSC[FPR] is set, the host controller sends resume signaling down the port. System software times the duration of the resume (nominally 20 milliseconds) then clears PORTSC[FPR]. When the host controller receives the write to transition PORTSC[FPR] to zero, it completes the resume sequence as defined in the USB specification, and clears both PORTSC[FPR] and PORTSC[SUSP]. Software-initiated port resumes do not affect the port change detect bit (USBSTS[PCI]) nor do they cause an interrupt if USBINTR[PCE] (port change interrupt enable) is a one. When a wake event occurs on a suspended port, the resume signaling is detected by the port and the resume is reflected downstream within 100 µsec. The port's PORTSC[FPR] bit is set and USBSTS[PCI] is set. If USBINTR[PCE] is a one, the host controller issues a hardware interrupt.

System software observes the resume event on the port, delays a port resume time (nominally 20 milliseconds), then terminates the resume sequence by clearing PORTSC[FPR] in the port. The host controller receives the write of zero to PORTSC[FPR], terminates the resume sequence and clears PORTSC[FPR] and PORTSC[SUSP]. Software can determine that the port is enabled (not suspended) by sampling the PORTSC register and observing that the SUSP and FPR bits are zero. Software must ensure that the host controller is running (that is, USBSTS[HCH] is a zero), before terminating a resume by clearing the port's PORTSC[FPR] bit. If HCH is a one when PORTSC[FPR] is cleared, then SOFs will not occur down the enabled port and the device will return to suspend mode in a maximum of 10 milliseconds.

Table 21-63 summarizes the wake-up events. Whenever a resume event is detected, USBSTS[PCI] is set. If USBINTR[PCE] (port change interrupt enable) is a one, the host controller also generates an interrupt on the resume event. Software acknowledges the resume event interrupt by clearing the USBSTS[PCI].

**Table 21-63. Behavior During Wake-up Events**

| Port Status and Signaling Type | Signaled Port Response | Device State | |
|---|---|---|---|
| | | D0 | not D0 |
| Port disabled, resume K-State received | No effect | N/A | N/A |
| Port suspended, Resume K-State received | Resume reflected downstream on signaled port. PORTSC[FPR] is set. USBSTS[PCI] is set. | [1], [2] | [2] |
| Port is enabled, disabled or suspended, and the port's WKDSCNNT_E bit, PORTSC[WKDS], is set. A disconnect is detected. | Depending on the initial port state, the PORTSC Connect (CCS) and Enable (PE) status bits are cleared, and the Connect Change status bit (CSC) is set. USBSTS[PCI] is set. | [1], [2] | [2] |
| Port is enabled, disabled or suspended, and the port's WKDSCNNT_E bit, PORTSC[WKDS], is cleared. A disconnect is detected. | Depending on the initial port state, the PORTSC Connect (CCS) and Enable (PE) status bits are cleared, and the Connect Change status bit (CSC) is set. USBSTS[PCI] is set. | [1], [3] | [3] |
| Port is not connected and the port's WKCNNT_E bit is a one. A connect is detected. | PORTSC Connect Status (CCS) and Connect Status Change (CSC) bits are set. USBSTS[PCI] is set. | [1], [2] | [2] |
| Port is not connected and the port's WKCNNT_E bit is a zero. A connect is detected. | PORTSC Connect Status (CCS) and Connect Status Change (CSC) bits are set. USBSTS[PCI] is set. | [1], [3] | [3] |
| Port is connected and the port's WKOC_E bit is a one. An over-current condition occurs. | PORTSC Over-current Active (OCA), Over-current Change (OCC) bits are set. If Port Enable/Disable bit (PE) is a one, it is cleared. USBSTS[PCI] is set | [1], [2] | [2] |
| Port is connected and the port's WKOC_E bit is a zero. An over-current condition occurs. | PORTSC Over-current Active (OCA), Over-current Change (OCC) bits are set. If Port Enable/Disable bit (PE) is a one, it is cleared. USBSTS[PCI] is set. | [1], [3] | [3] |

[1] Hardware interrupt issued if USBINTR[PCE] (port change interrupt enable) is set.

[2] PME# asserted if enabled (Note: PME Status must always be set).

[3] PME# not asserted.

### 21.6.5 Schedule Traversal Rules

The host controller executes transactions for devices using a simple, shared-memory schedule. The schedule is comprised of a few data structures, organized into two distinct lists. The data structures are designed to provide the maximum flexibility required by USB, minimize memory traffic and hardware/software complexity.

System software maintains two schedules for the host controller: a periodic schedule and an asynchronous schedule. The root of the periodic schedule is the PERIODICLISTBASE register. See Section 21.3.2.6, “Periodic Frame List Base Address Register (PERIODICLISTBASE),” for more information. The PERIODICLISTBASE register is the physical memory base address of the periodic frame list. The periodic frame list is an array of physical memory pointers. The objects referenced from the frame list must be valid schedule data structures as defined in Section 21.5, “Host Data Structures.” In each micro-frame, if the periodic schedule is enabled (see) then the host controller must execute from the periodic schedule before executing from the asynchronous schedule. It will only execute from the asynchronous schedule after it encounters the end of the periodic schedule. The host controller traverses the periodic schedule by constructing an array offset reference from the PERIODICLISTBASE and the FRINDEX registers (see Figure 21-42). It fetches the element and begins traversing the graph of linked schedule data structures.

The end of the periodic schedule is identified by a next link pointer of a schedule data structure having its T-bit set. When the host controller encounters a T-Bit set during a horizontal traversal of the periodic list, it interprets this as an End-Of-Periodic-List mark. This causes the host controller to cease working on the periodic schedule and transitions immediately to traversing the asynchronous schedule. Once this transition is made, the host controller executes from the asynchronous schedule until the end of the micro-frame.

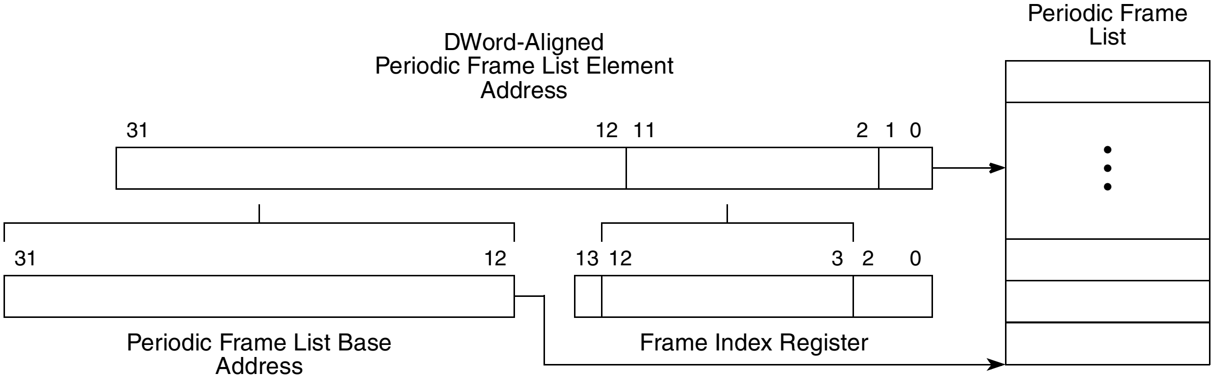

Periodic Frame List
DWord-Aligned Periodic Frame List Element Address
31
12
11
2
1
0
31
12
13
12
3
2
0
Periodic Frame List Base Address
Frame Index Register

**Figure 21-42. Derivation of Pointer into Frame List Array**

When the host controller determines that it is time to execute from the asynchronous list, it uses the operational register ASYNCLISTADDR to access the asynchronous schedule, as shown in Figure 21-43.

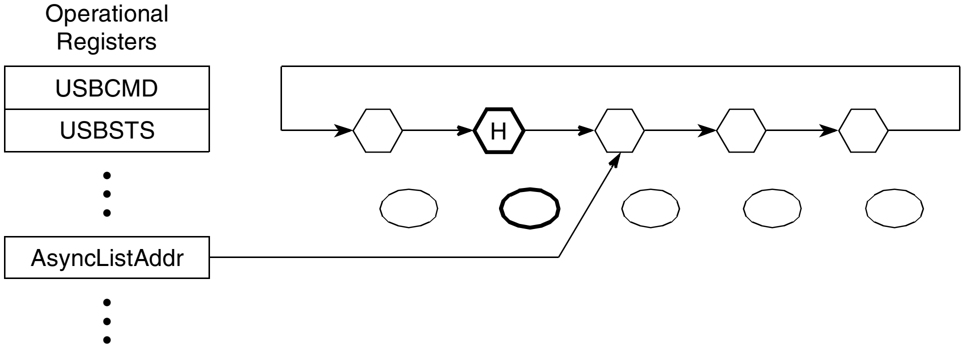


Figure 21-43. General Format of Asynchronous Schedule List

The ASYNCLISTADDR register contains a physical memory pointer to the next queue head. When the host controller makes a transition to executing the asynchronous schedule, it begins by reading the queue head referenced by the ASYNCLISTADDR register. Software must set queue head horizontal pointer T-bits to a zero for queue heads in the asynchronous schedule.

## 21.6.6 Periodic Schedule Frame Boundaries vs. Bus Frame Boundaries

The USB Specification Revision 2.0 requires that the frame boundaries (SOF frame number changes) of the high-speed bus and the full- and low-speed bus(es) below USB 2.0 hubs be strictly aligned. Super-imposed on this requirement is that USB 2.0 hubs manage full- and low-speed transactions via a micro-frame pipeline (see start- (SS) and complete- (CS) splits illustrated in Figure 21-44). A simple, direct projection of the frame boundary model into the host controller interface schedule architecture creates tension (complexity for both hardware and software) between the frame boundaries and the scheduling mechanisms required to service the full- and low-speed transaction translator periodic pipelines.

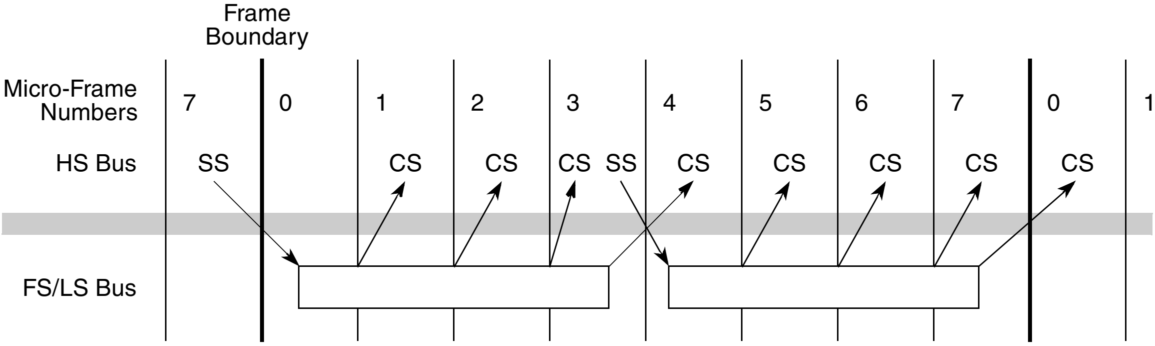


Figure 21-44. Frame Boundary Relationship Between HS Bus and FS/LS Bus

The simple projection, as Figure 21-44 illustrates, introduces frame-boundary wrap conditions for scheduling on both the beginning and end of a frame. In order to reduce the complexity for hardware and software, the host controller is required to implement a one micro-frame phase shift for its view of frame boundaries. The phase shift eliminates the beginning of frame and frame-wrap scheduling boundary conditions.

The implementation of this phase shift requires that the host controller use one register value for accessing the periodic frame list and another value for the frame number value included in the SOF token. These two values are separate, but tightly coupled. The periodic frame list is accessed via the Frame List Index Register (FRINDEX). Bits FRINDEX[2–0], represent the micro-frame number. The SOF value is coupled to the value of FRINDEX[13–3]. Both FRINDEX[13–3] and the SOF value are incremented based on FRINDEX[2–0]. It is required that the SOF value be delayed from the FRINDEX value by one micro-frame. The one micro-frame delay yields a host controller periodic schedule and bus frame boundary relationship as illustrated in Figure 21-45. This adjustment allows software to trivially schedule the periodic start and complete-split transactions for full-and low-speed periodic endpoints, using the natural alignment of the periodic schedule interface.

Figure 21-45 illustrates how periodic schedule data structures relate to schedule frame boundaries and bus frame boundaries. To aid the presentation, two terms are defined. The host controller's view of the 1-millisecond boundaries is called H-Frames. The high-speed bus's view of the 1-millisecond boundaries is called B-Frames.

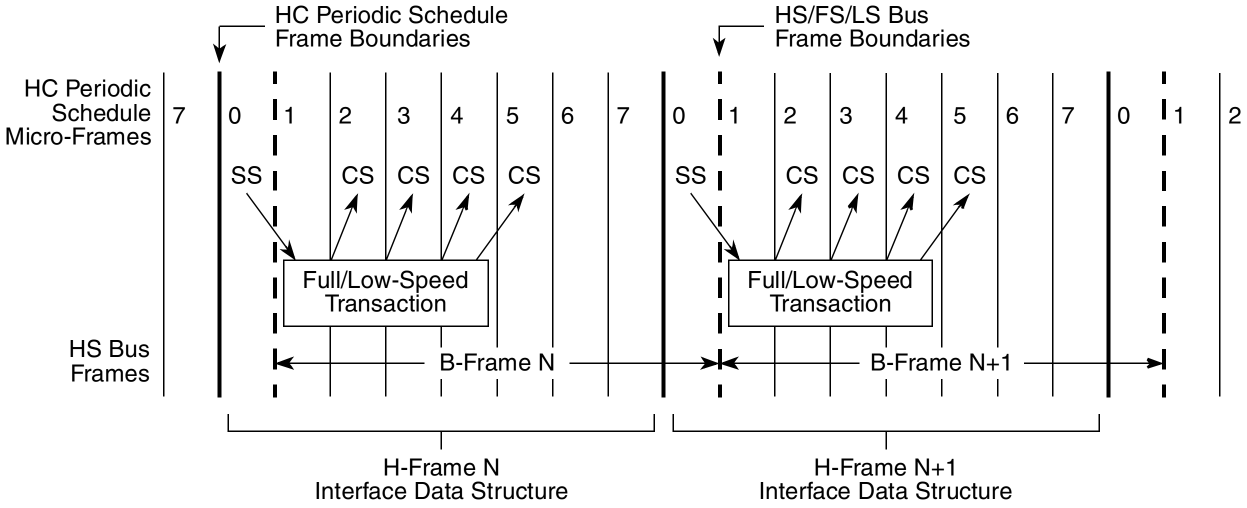


**Figure 21-45. Relationship of Periodic Schedule Frame Boundaries to Bus Frame Boundaries**

H-Frame boundaries for the host controller correspond to increments of FRINDEX[13–3]. Micro-frame numbers for the H-Frame are tracked by FRINDEX[2–0]. B-Frame boundaries are visible on the high-speed bus via changes in the SOF token's frame number. Micro-frame numbers on the high-speed bus are only derived from the SOF token's frame number (that is, the high-speed bus will see eight SOFs with the same frame number value). H-Frames and B-Frames have the fixed relationship (that is, B-Frames lag H-Frames by one micro-frame time) illustrated in Figure 21-45. The host controller's periodic schedule is naturally aligned to H-Frames. Software schedules transactions for full- and low-speed periodic endpoints relative the H-Frames. The result is these transactions execute on the high-speed bus at exactly the right time for the USB 2.0 hub periodic pipeline. As described in Section 21.3.2.4, “Frame Index Register (FRINDEX),” the SOF Value can be implemented as a shadow register (in this example, called SOFV), which lags the FRINDEX register bits [13–3] by one micro-frame count. Table 21-64 illustrates the required relationship between the value of FRINDEX and the value of SOFV. This lag behavior can be accomplished by incrementing FRINDEX[13–3] based on carry-out on the 7 to 0 increment of FRINDEX[2–0] and incrementing SOFV based on the transition of 0 to 1 of FRINDEX[2–0].

Software is allowed to write to FRINDEX. Section 21.3.2.4, “Frame Index Register (FRINDEX),” provides the requirements that software should adhere when writing a new value in FRINDEX.

**Table 21-64. Operation of FRINDEX and SOFV (SOF Value Register)**

| Current | | | Next | | |
|---|---|---|---|---|---|
| **FRINDEX[13–3]** | **SOFV** | **FRINDEX[2–0]** | **FRINDEX[13–3]** | **SOFV** | **FRINDEX[2–0]** |
| N | N | 111 | N+1 | N | 000 |
| N+1 | N | 000 | N+1 | N+1 | 001 |
| N+1 | N+1 | 001 | N+1 | N+1 | 010 |
| N+1 | N+1 | 010 | N+1 | N+1 | 011 |
| N+1 | N+1 | 011 | N+1 | N+1 | 100 |
| N+1 | N+1 | 100 | N+1 | N+1 | 101 |
| N+1 | N+1 | 101 | N+1 | N+1 | 110 |
| N+1 | N+1 | 110 | N+1 | N+1 | 111 |

### 21.6.7 Periodic Schedule

The periodic schedule traversal is enabled or disabled through USBCMD[PSE] (periodic schedule enable). If USBCMD[PSE] is cleared, then the host controller simply does not try to access the periodic frame list via the PERIODICLISTBASE register. Likewise, when USBCMD[PSE] is a one, then the host controller does use the PERIODICLISTBASE register to traverse the periodic schedule. The host controller will not react to modifications to USBCMD[PSE] immediately. In order to eliminate conflicts with split transactions, the host controller evaluates USBCMD[PSE] only when FRINDEX[2–0] is zero. System software must not disable the periodic schedule if the schedule contains an active split transaction work item that spans the 0b000 micro-frame. These work items must be removed from the schedule before USBCMD[PSE] is cleared. USBSTS[PS] (periodic schedule status) indicates status of the periodic schedule. System software enables (or disables) the periodic schedule by setting (or clearing) USBCMD[PSE]. Software then can poll USBSTS[PS] to determine when the periodic schedule has made the desired transition. Software must not modify USBCMD[PSE] unless the value of USBCMD[PSE] equals that of USBSTS[PS].

The periodic schedule is used to manage all isochronous and interrupt transfer streams. The base of the periodic schedule is the periodic frame list. Software links schedule data structures to the periodic frame list to produce a graph of scheduled data structures. The graph represents an appropriate sequence of transactions on the USB. Figure 21-46 illustrates isochronous transfers (using iTDs and siTDs) with a period of one are linked directly to the periodic frame list. Interrupt transfers (are managed with queue heads) and isochronous streams with periods other than one are linked following the period-one iTD/siTDs. Interrupt queue heads are linked into the frame list ordered by poll rate. Longer poll rates are linked first (for example, closest to the periodic frame list), followed by shorter poll rates, with queue heads with a poll rate of one, on the very end.

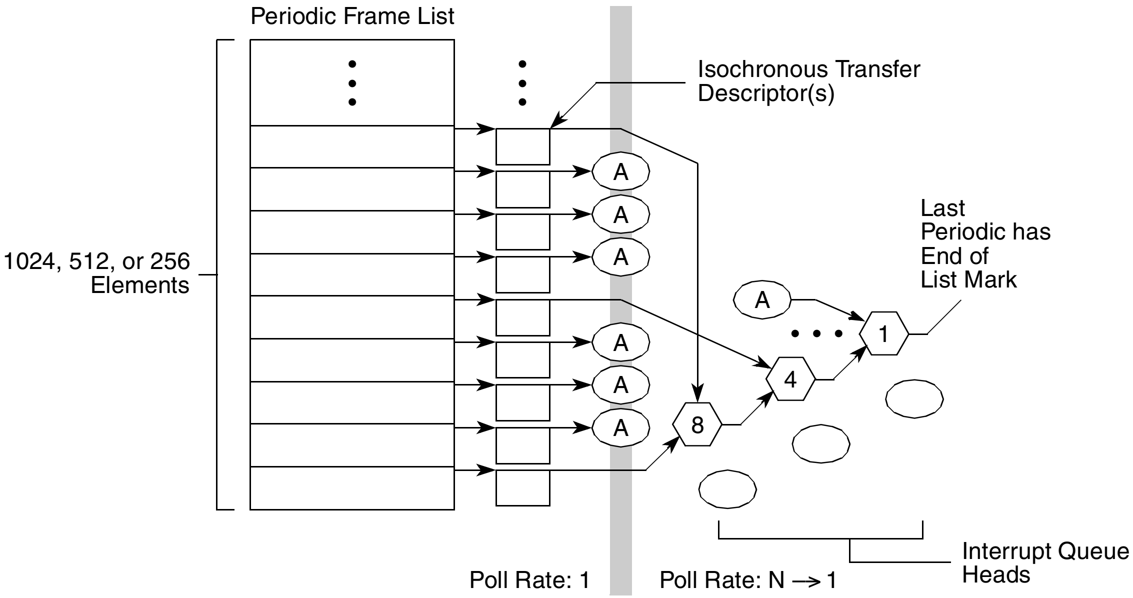

Periodic Frame List
1024, 512, or 256 Elements
Isochronous Transfer Descriptor(s)
A
A
A
A
A
A
A
8
4
1
Last Periodic has End of List Mark
Interrupt Queue Heads
Poll Rate: 1
Poll Rate: N —> 1

**Figure 21-46. Example Periodic Schedule**

### 21.6.8 Managing Isochronous Transfers Using iTDs

The structure of an iTD is presented in Isochronous (High-Speed) Transfer Descriptor (iTD). There are four distinct sections to an iTD:

- The first field is the Next Link Pointer. This field is for schedule linkage purposes only.
- Transaction description array. This area is an eight-element array. Each element represents control and status information for one micro-frame's worth of transactions for a single high-speed isochronous endpoint.
- The buffer page pointer array is a 7-element array of physical memory pointers to data buffers. These are 4K aligned pointers to physical memory.
- Endpoint capabilities. This area utilizes the unused low-order 12 bits of the buffer page pointer array. The fields in this area are used across all transactions executed for this iTD, including endpoint addressing, transfer direction, maximum packet size and high-bandwidth multiplier.

#### 21.6.8.1 Host Controller Operational Model for iTDs

The host controller uses FRINDEX register bits 12–3 to index into the periodic frame list. This means that the host controller visits each frame list element eight consecutive times before incrementing to the next periodic frame list element. Each iTD contains eight transaction descriptions, which map directly to FRINDEX register bits 2–0. Each iTD can span 8 micro-frames worth of transactions. When the host controller fetches an iTD, it uses FRINDEX register bits 2–0 to index into the transaction description array. When the first iTD in the periodic list is traversed after periodic schedule is enabled, the value of FRINDEX[2:0] may be other then 0, so the first transaction issued by the controller may be any of the eight available active transactions. If the active bit in the Status field of the indexed transaction description is cleared, the host controller ignores the iTD and follows the Next pointer to the next schedule data structure.

When the indexed active bit is a one the host controller continues to parse the iTD. It stores the indexed transaction description and the general endpoint information (device address, endpoint number, maximum packet size, etc.). It also uses the Page Select (PG) field to index the buffer pointer array, storing the selected buffer pointer and the next sequential buffer pointer. For example, if PG field is a 0, then the host controller will store Page 0 and Page 1.

The host controller constructs a physical data buffer address by concatenating the current buffer pointer (as selected using the current transaction description's PG field) and the transaction description's Transaction Offset field. The host controller uses the endpoint addressing information and I/O-bit to execute a transaction to the appropriate endpoint. When the transaction is complete, the host controller clears the active bit and writes back any additional status information to the Status field in the currently selected transaction description.

The data buffer associated with the iTD must be virtually contiguous memory. Seven page pointers are provided to support eight high-bandwidth transactions regardless of the starting packet’s offset alignment into the first page. A starting buffer pointer (physical memory address) is constructed by concatenating the page pointer (example: page 0 pointer) selected by the active transaction descriptions’ PG (example value: 0b00) field with the transaction offset field. As the transaction moves data, the host controller must detect when an increment of the current buffer pointer will cross a page boundary. When this occurs the host controller simply replaces the current buffer pointer’s page portion with the next page pointer (example: page 1 pointer) and continues to move data. The size of each bus transaction is determined by the value in the Maximum Packet Size field. An iTD supports high-bandwidth pipes via the Mult (multiplier) field. When the Mult field is 1, 2, or 3, the host controller executes the specified number of Maximum Packet sized bus transactions for the endpoint in the current micro-frame. In other words, the Mult field represents a transaction count for the endpoint in the current micro-frame. If the Mult field is zero, the operation of the host controller is undefined. The transfer description is used to service all transactions indicated by the Mult field.

For OUT transfers, the value of the Transaction *n* Length field represents the total bytes to be sent during the micro-frame. The Mult field must be set by software to be consistent with Transaction *n* Length and Maximum Packet Size. The host controller will send the bytes in Maximum Packet Sized portions. After each transaction, the host controller decrements it's local copy of Transaction *n* Length by Maximum Packet Size. The number of bytes the host controller sends is always Maximum Packet Size or Transaction *n* Length, whichever is less. The host controller advances the transfer state in the transfer description, updates the appropriate record in the iTD and moves to the next schedule data structure. The maximum sized transaction supported is 3 × 1024 bytes.

For IN transfers, the host controller issues Mult transactions. It is assumed that software has properly initialized the iTD to accommodate all of the possible data. During each IN transaction, the host controller must use Maximum Packet Size to detect packet babble errors. The host controller keeps the sum of bytes received in the Transaction *n* Length field. After all transactions for the endpoint have completed for the micro-frame, Transaction *n* Length contains the total bytes received. If the final value of Transaction *n* Length is less than the value of Maximum Packet Size, then less data than was allowed for was received from the associated endpoint. This short packet condition does not set USBSTS[UI] (USB interrupt). The host controller will not detect this condition. If the device sends more than Transaction *n* Length or Maximum Packet Size bytes (whichever is less), then the host controller will set the Babble Detected bit and clear the Active bit. Note, that the host controller is not required to update the iTD field Transaction *n*

Length in this error scenario. If the Mult field is greater than one, then the host controller will automatically execute the value of Mult transactions. The host controller will not execute all Mult transactions if:

- The endpoint is an OUT and Transaction *n* Length goes to zero before all the Mult transactions have executed (ran out of data), or
- The endpoint is an IN and the endpoint delivers a short packet, or an error occurs on a transaction before Mult transactions have been executed. The end of micro-frame may occur before all of the transaction opportunities have been executed. When this happens, the transfer state of the transfer description is advanced to reflect the progress that was made, the result written back to the iTD and the host controller proceeds to processing the next micro-frame.

#### 21.6.8.2 Software Operational Model for iTDs

A client buffer request to an isochronous endpoint may span 1 to N micro-frames. When N is larger than one, system software may have to use multiple iTDs to read or write data with the buffer (if N is larger than eight, it must use more than one iTD).

Figure 21-47 illustrates the simple model of how a client buffer is mapped by system software to the periodic schedule (that is, the periodic frame list and a set of iTDs). On the right is the client description of its request. The description includes a buffer base address plus additional annotations to identify which portions of the buffer should be used with each bus transaction. In the middle is the iTD data structures used by the system software to service the client request. Each iTD can be initialized to service up to 24 transactions, organized into eight groups of up to three transactions each. Each group maps to one micro-frame's worth of transactions. The EHCI controller does not provide per-transaction results within a micro-frame. It treats the per-micro-frame transactions as a single logical transfer. On the left is the host controller’s frame list. System software establishes references from the appropriate locations in the frame list to each of the appropriate iTDs. If the buffer is large, then system software can use a small set of iTDs to service the entire buffer. System software can activate the transaction description records (contained in each iTD) in any pattern required for the particular data stream.

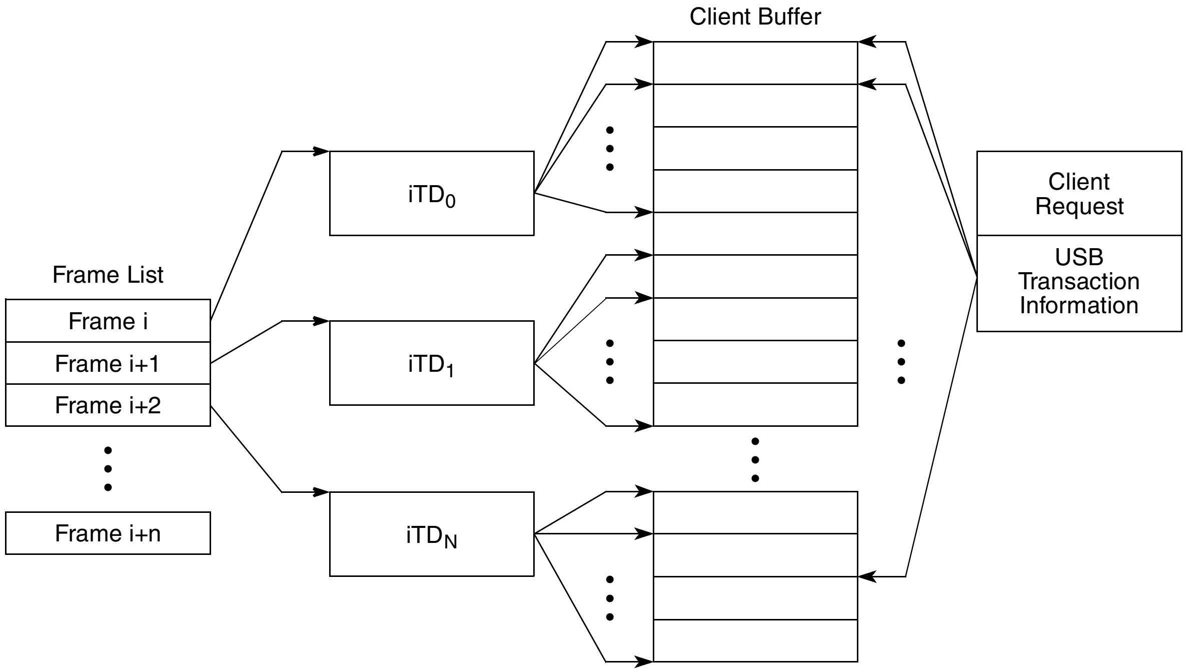


**Figure 21-47. Example Association of iTDs to Client Request Buffer**

As noted above, the client request includes a pointer to the base of the buffer and offsets into the buffer to annotate which buffer sections are to be used on each bus transaction that occurs on this endpoint. System software must initialize each transaction description in an iTD to ensure it uses the correct portion of the client buffer. For example, for each transaction description, the PG field is set to index the correct physical buffer page pointer and the Transaction Offset field is set relative to the correct buffer pointer page (for example, the same one referenced by the PG field). When the host controller executes a transaction it selects a transaction description record based on FRINDEX[2–0]. It then uses the current Page Buffer Pointer (as selected by the PG field) and concatenates to the transaction offset field. The result is a starting buffer address for the transaction. As the host controller moves data for the transaction, it must watch for a page wrap condition and properly advance to the next available Page Buffer Pointer. System software must not use the Page 6 buffer pointer in a transaction description where the length of the transfer will wrap a page boundary. Doing so yields undefined behavior. The host controller hardware is not required to alias the page selector to page zero. USB 2.0 isochronous endpoints can specify a period greater than one. Software can achieve the appropriate scheduling by linking iTDs into the appropriate frames (relative to the frame list) and by setting appropriate transaction description elements active bits to a one.

### 21.6.8.2.1 Periodic Scheduling Threshold

The Isochronous Scheduling Threshold field in the HCCPARAMS capability register is an indicator to system software as to how the host controller pre-fetches and effectively caches schedule data structures. It is used by system software when adding isochronous work items to the periodic schedule. The value of this field indicates to system software the minimum distance it can update isochronous data (relative to the current location of the host controller execution in the periodic list) and still have the host controller process them.

The iTD and siTD data structures each describe 8 micro-frames worth of transactions. The host controller is allowed to cache one (or more) of these data structures in order to reduce memory traffic. There are three basic caching models that account for the fact the isochronous data structures span 8 micro-frames. The three caching models are: no caching, micro-frame caching and frame caching.

When software is adding new isochronous transactions to the schedule, it always performs a read of the FRINDEX register to determine the current frame and micro-frame the host controller is currently executing. Of course, there is no information about where in the micro-frame the host controller is, so a constant uncertainty factor of one micro-frame has to be assumed. Combining the knowledge of where the host controller is executing with the knowledge of the caching model allows the definition of simple algorithms for how closely software can reliably work to the executing host controller.

No caching is indicated with a value of zero in the Isochronous Scheduling Threshold field. The host controller may pre-fetch data structures during a periodic schedule traversal (per micro-frame) but will always dump any accumulated schedule state at the end of the micro-frame. At the appropriate time relative to the beginning of every micro-frame, the host controller always begins schedule traversal from the frame list. Software can use the value of the FRINDEX register (plus the constant 1 uncertainty-factor) to determine the approximate position of the executing host controller. When no caching is selected, software can add an isochronous transaction as near as 2 micro-frames in front of the current executing position of the host controller.

Frame caching is indicated with a non-zero value in bit [7] of the Isochronous Scheduling Threshold field. In the frame-caching model, system software assumes that the host controller caches one (or more) isochronous data structures for an entire frame (8 micro-frames). Software uses the value of the FRINDEX register (plus the constant 1 uncertainty) to determine the current micro-frame/frame (assume modulo 8 arithmetic in adding the constant 1 to the micro-frame number). For any current frame N, if the current micro-frame is 0 to 6, then software can safely add isochronous transactions to Frame N + 1. If the current micro-frame is 7, then software can add isochronous transactions to Frame N + 2.

Micro-frame caching is indicated with a non-zero value in the least-significant 3 bits of the Isochronous Scheduling Threshold field. System software assumes the host controller caches one or more periodic data structures for the number of micro-frames indicated in the Isochronous Scheduling Threshold field. For example, if the count value were 2, then the host controller keeps a window of 2 micro-frames worth of state (current micro-frame, plus the next) on-chip. On each micro-frame boundary, the host controller releases the current micro-frame state and begins accumulating the next micro-frame state.

### 21.6.9 Asynchronous Schedule

The asynchronous schedule traversal is enabled or disabled through USBCMD[ASE] (asynchronous schedule enable). If USBCMD[ASE] is cleared, then the host controller simply does not try to access the asynchronous schedule via the ASYNCLISTADDR register. Likewise, if USBCMD[ASE] is set, the host controller does use the ASYNCLISTADDR register to traverse the asynchronous schedule. Modifications to USBCMD[ASE] are not necessarily immediate. Rather the new value of the bit will only be taken into consideration the next time the host controller needs to use the value of the ASYNCLISTADDR register to get the next queue head.

USBSTS[AS] indicates status of the asynchronous schedule. System software enables (or disables) the asynchronous schedule by writing a one (or zero) to USBCMD[ASE]. Software then can poll

USBSTS[AS] to determine when the asynchronous schedule has made the desired transition. Software must not modify USBCMD[ASE] unless the value of USBCMD[ASE] equals that of the USBSTS[AS] (asynchronous schedule status).

The asynchronous schedule is used to manage all Control and Bulk transfers. Control and Bulk transfers are managed using queue head data structures. The asynchronous schedule is based at the ASYNCLISTADDR register. The default value of the ASYNCLISTADDR register after reset is undefined and the schedule is disabled when USBCMD[ASE] is cleared.

Software may only write this register with defined results when the schedule is disabled, for example, USBCMD[ASE] and the USBSTS[AS] are cleared. System software enables execution from the asynchronous schedule by writing a valid memory address (of a queue head) into this register. Then software enables the asynchronous schedule by setting USBCMD[ASE]. The asynchronous schedule is actually enabled when USBSTS[AS] is set.

When the host controller begins servicing the asynchronous schedule, it begins by using the value of the ASYNCLISTADDR register. It reads the first referenced data structure and begins executing transactions and traversing the linked list as appropriate. When the host controller completes processing the asynchronous schedule, it retains the value of the last accessed queue head's horizontal pointer in the ASYNCLISTADDR register. Next time the asynchronous schedule is accessed, this is the first data structure that is serviced. This provides round-robin fairness for processing the asynchronous schedule.

A host controller completes processing the asynchronous schedule when one of the following events occur:

- The end of a micro-frame occurs.
- The host controller detects an empty list condition
- The schedule has been disabled through USBCMD[ASE].

The queue heads in the asynchronous list are linked into a simple circular list as shown in Figure 21-43. Queue head data structures are the only valid data structures that may be linked into the asynchronous schedule. An isochronous transfer descriptor (iTD or siTD) in the asynchronous schedule yields undefined results.

The maximum packet size field in a queue head is sized to accommodate the use of this data structure for all non-isochronous transfer types. The USB Specification, Revision 2.0 specifies the maximum packet sizes for all transfer types and transfer speeds. System software should always parameterize the queue head data structures according to the core specification requirements.

### 21.6.9.1 Adding Queue Heads to Asynchronous Schedule

This is a software requirement section. There are two independent events for adding queue heads to the asynchronous schedule. The first is the initial activation of the asynchronous list. The second is inserting a new queue head into an activated asynchronous list.

Activation of the list is simple. System software writes the physical memory address of a queue head into the ASYNCLISTADDR register, then enables the list by setting USBCMD[ASE] to a one.

When inserting a queue head into an active list, software must ensure that the schedule is always coherent from the host controllers' point of view. This means that the system software must ensure that all queue head pointer fields are valid. For example qTD pointers have T-Bits set or reference valid qTDs and the Horizontal Pointer references a valid queue head data structure. The following algorithm represents the functional requirements:

```
InsertQueueHead (pQHeadCurrent, pQueueHeadNew)
    --
    -- Requirement: all inputs must be properly initialized.
    --
    -- pQHeadCurrent is a pointer to a queue head that is
    -- already in the active list
    -- pQHeadNew is a pointer to the queue head to be added
    --
    -- This algorithm links a new queue head into a existing
    -- list
    --
    pQueueHeadNew.HorizontalPointer = pQueueHeadCurrent.HorizontalPointer
    pQueueHeadCurrent.HorizontalPointer = physicalAddressOf(pQueueHeadNew)
End InsertQueueHead
```

### 21.6.9.2 Removing Queue Heads from Asynchronous Schedule

This is a software requirement section. There are two independent events for removing queue heads from the asynchronous schedule. The first is shutting down (deactivating) the asynchronous list. The second is extracting a single queue head from an activated list. Software deactivates the asynchronous schedule by setting USBCMD[ASE] to a zero. Software can determine when the list is idle when USBSTS[AS] is cleared. The normal mode of operation is that software removes queue heads from the asynchronous schedule without shutting it down. Software must not remove an active queue head from the schedule. Software should first deactivate all active qTDs, wait for the queue head to go inactive, then remove the queue head from the asynchronous list. Software removes a queue head from the asynchronous list using the following algorithm. Software merely must ensure all of the link pointers reachable by the host controller are kept consistent.

```
UnlinkQueueHead (pQHeadPrevious, pQueueHeadToUnlink, pQHeadNext)
    --
    -- Requirement: all inputs must be properly initialized.
    --
    -- pQHeadPrevious is a pointer to a queue head that
    -- references the queue head to remove
    -- pQHeadToUnlink is a pointer to the queue head to be
    -- removed
    -- pQheadNext is a pointer to a queue head still in the
    -- schedule. Software provides this pointer with the
    -- following strict rules:
    -- if the host software is one queue head, then
    -- pQHeadNext must be the same as
    -- QueueheadToUnlink.HorizontalPointer. If the host
    -- software is unlinking a consecutive series of
    -- queue heads, QHeadNext must be set by software to
    -- the queue head remaining in the schedule.
    --
    -- This algorithm unlinks a queue head from a circular list
    --
    pQueueHeadPrevious.HorizontalPointer = pQueueHeadToUnlink.HorizontalPointer
    pQueueHeadToUnlink.HorizontalPointer = pQHeadNext
End UnlinkQueueHead
```

If software removes the queue head with the H-bit set, it must select another queue head still linked into the schedule and set its H-bit. This should be completed before removing the queue head. The requirement is that software keep one queue head in the asynchronous schedule, with its H-bit set. At the point software has removed one or more queue heads from the asynchronous schedule, it is unknown whether the host controller has a cached pointer to them. Similarly, it is unknown how long the host controller might retain the cached information, as it is implementation dependent and may be affected by the actual dynamics of the schedule load. Therefore, once software has removed a queue head from the asynchronous list, it must retain the coherency of the queue head (link pointers). It cannot disturb the removed queue heads until it knows that the host controller does not have a local copy of a pointer to any of the removed data structures.

The method software uses to determine when it is safe to modify a removed queue head is to handshake with the host controller. The handshake mechanism allows software to remove items from the asynchronous schedule, then execute a simple, lightweight handshake that is used by software as a key that it can free (or reuse) the memory associated the data structures it has removed from the asynchronous schedule.

The handshake is implemented with three bits in the host controller. The first bit is a command bit (USBCMD[IAA]—interrupt on async advance doorbell) that allows software to inform the host controller that something has been removed from its asynchronous schedule. The second bit is a status bit (USBSTS[AAI]—interrupt on async advance) that the host controller sets after it has released all on-chip state that may potentially reference one of the data structures just removed. When the host controller sets this status bit, it also clears the command bit. The third bit is an interrupt enable (USBINTR[AAE]—interrupt on async advance enable) that is matched with the status bit. If the status bit is set and the interrupt enable bit is set, then the host controller asserts a hardware interrupt.

Figure 21-48 illustrates a general example where consecutive queue heads (B and C) are unlinked from the schedule using the algorithm above. Before the unlink operation, the host controller has a copy of queue head A.

The unlink algorithm requires that as software unlinks each queue head, the unlinked queue head is loaded with the address of a queue head that will remain in the asynchronous schedule.

When the host controller observes that doorbell bit being set, it makes a note of the local reachable schedule information. In this example, the local reachable schedule information includes both queue heads (A & B). It is sufficient that the host controller can set the status bit (and clear the doorbell bit) as soon as it has traversed beyond current reachable schedule information (that is, traversed beyond queue head (B) in this example).

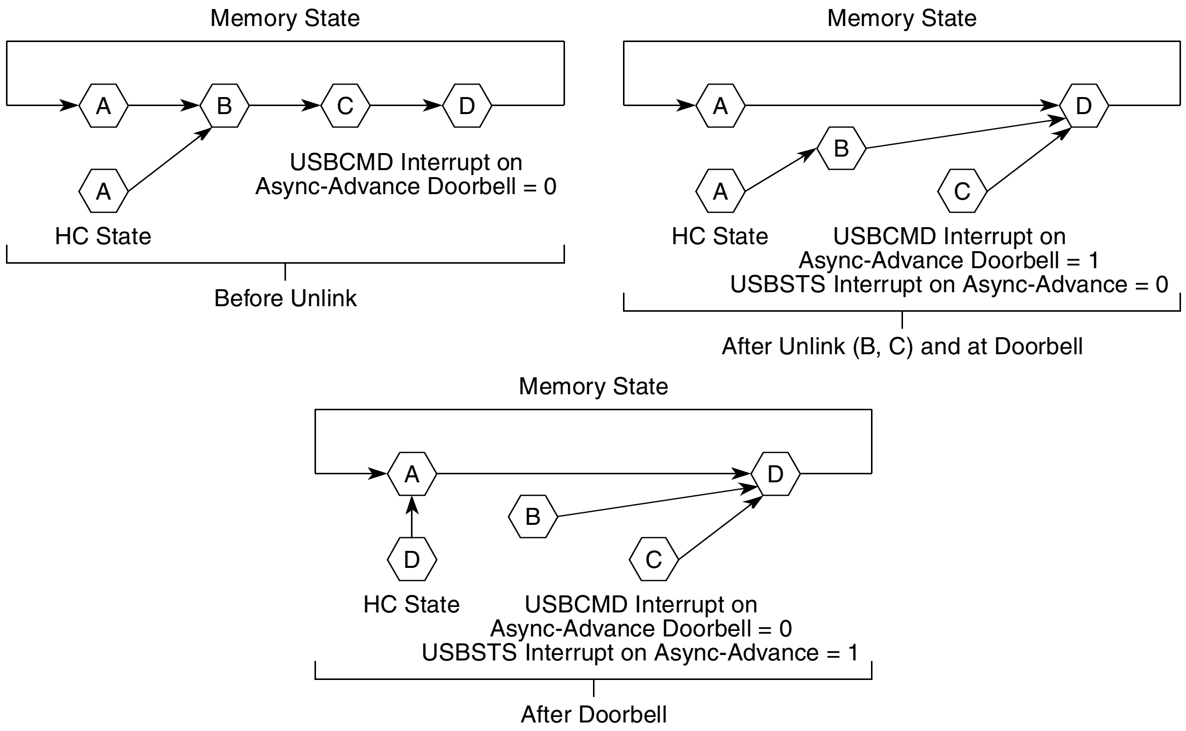

Memory State
A
B
C
D
A
HC State
USBCMD Interrupt on
Async-Advance Doorbell = 0
Before Unlink
Memory State
A
D
B
A
C
HC State
USBCMD Interrupt on
Async-Advance Doorbell = 1
USBSTS Interrupt on Async-Advance = 0
After Unlink (B, C) and at Doorbell
Memory State
A
D
B
D
C
HC State
USBCMD Interrupt on
Async-Advance Doorbell = 0
USBSTS Interrupt on Async-Advance = 1
After Doorbell

**Figure 21-48. Generic Queue Head Unlink Scenario**

Alternatively, a host controller implementation is allowed to traverse the entire asynchronous schedule list (for example, observed the head of the queue (twice)) before setting USBSTS[AAI].

Software may re-use the memory associated with the removed queue heads after it observes USBSTS[AAI] is set, following assertion of the doorbell. Software should acknowledge the interrupt on async advance status as indicated in the USBSTS register, before using the doorbell handshake again

### 21.6.9.3 Empty Asynchronous Schedule Detection

EHCI uses two bits to detect when the asynchronous schedule is empty. The queue head data structure (see Figure 21-40) defines an H-bit in the queue head, which allows software to mark a queue head as being the head of the reclaim list. host controller also keeps a 1-bit flag in the USBSTS register (Reclamation) that is cleared when the host controller observes a queue head with the H-bit set. The reclamation flag in the status register is set when any USB transaction from the asynchronous schedule is executed (or whenever the asynchronous schedule starts, see Section 21.6.9.4, “Asynchronous Schedule Traversal: Start Event.”

If the controller ever encounters an H-bit of one and a Reclamation bit of zero, the controller simply stops traversal of the asynchronous schedule.

An example illustrating the H-bit in a schedule is shown in Figure 21-49.

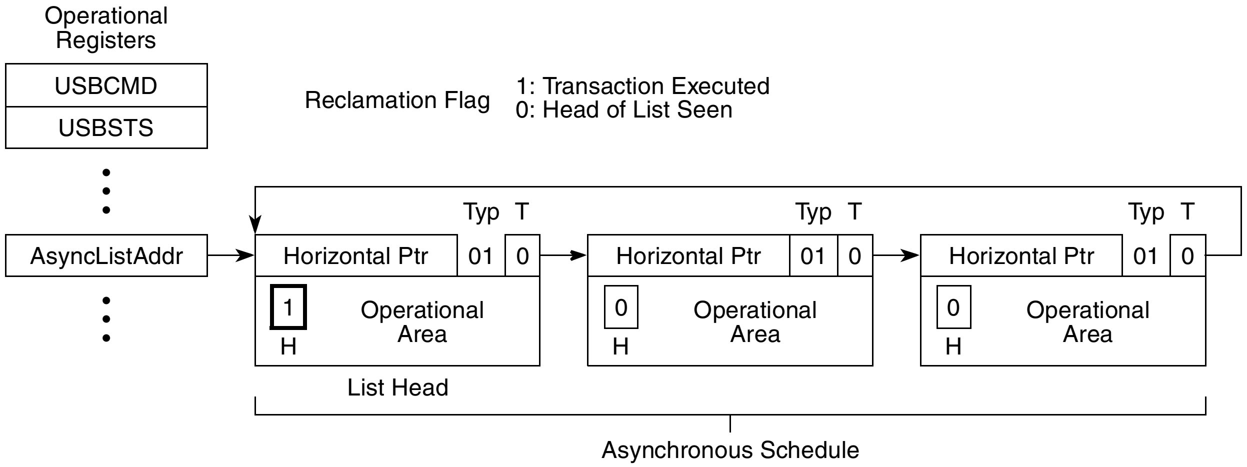


**Figure 21-49. Asynchronous Schedule List with Annotation to Mark Head of List**

### 21.6.9.4 Asynchronous Schedule Traversal: Start Event

Once the host controller has idled itself using the empty schedule detection, it naturally activates and begins processing from the Periodic Schedule at the beginning of each micro-frame. In addition, it may have idled itself early in a micro-frame. When this occurs (idles early in the micro-frame) the host controller must occasionally reactivate during the micro-frame and traverse the asynchronous schedule to determine whether any progress can be made. Asynchronous schedule Start Events are defined to be:

- Whenever the host controller transitions from the periodic schedule to the asynchronous schedule. If the periodic schedule is disabled and the asynchronous schedule is enabled, then the beginning of the micro-frame is equivalent to the transition from the periodic schedule, or
- The asynchronous schedule traversal restarts from a sleeping state.

### 21.6.9.5 Reclamation Status Bit (USBSTS Register)

The operation of the empty asynchronous schedule detection feature depends on the proper management of the Reclamation bit (RCL) in the USBSTS register. The host controller tests for an empty schedule just after it fetches a new queue head while traversing the asynchronous schedule. The host controller sets USBSTS[RCL] whenever an asynchronous schedule traversal Start Event occurs. USBSTS[RCL] is also set whenever the host controller executes a transaction while traversing the asynchronous schedule.The host controller clears USBSTS[RCL] whenever it finds a queue head with its H-bit set. Software should only set a queue head's H-bit if the queue head is in the asynchronous schedule. If software sets the H-bit in an interrupt queue head, the resulting behavior is undefined. The host controller may clear USBSTS[RCL] when executing from the periodic schedule.

## 21.6.10 Managing Control/Bulk/Interrupt Transfers via Queue Heads

This section presents an overview of how the host controller interacts with queuing data structures.

Queue heads use the Queue Element Transfer Descriptor (qTD) structure defined in Section 21.5.5, “Queue Element Transfer Descriptor (qTD).”

One queue head is used to manage the data stream for one endpoint. The queue head structure contains static endpoint characteristics and capabilities. It also contains a working area from where individual bus transactions for an endpoint are executed. Each qTD represents one or more bus transactions, which is defined in the context of the EHCI specification as a transfer.

The general processing model for the host controller's use of a queue head is simple:

- Read a queue head,
- Execute a transaction from the overlay area,
- Write back the results of the transaction to the overlay area
- Move to the next queue head.

If the host controller encounters errors during a transaction, the host controller will set one of the error reporting bits in the queue head's Status field. The Status field accumulates all errors encountered during the execution of a qTD (that is, the error bits in the queue head Status field are sticky until the transfer (qTD) has completed). This state is always written back to the source qTD when the transfer is complete. On transfer (for example, buffer or halt conditions) boundaries, the host controller must auto-advance (without software intervention) to the next qTD. Additionally, the hardware must be able to halt the queue so no additional bus transactions will occur for the endpoint and the host controller will not advance the queue.

### 21.6.10.1 Buffer Pointer List Use for Data Streaming with qTDs

A qTD has an array of buffer pointers, which is used to reference the data buffer for a transfer. The EHCI specification requires that the buffer associated with the transfer be virtually contiguous. This means that if the buffer spans more than one physical page, it must obey the following rules:

- The first portion of the buffer must begin at some offset in a page and extend through the end of the page.
- The remaining buffer cannot be allocated in small chunks scattered around memory. For each 4K chunk beyond the first page, each buffer portion matches to a full 4K page. The final portion, which may only be large enough to occupy a portion of a page, must start at the top of the page and be contiguous within that page.

Figure 21-50 illustrates these requirements.

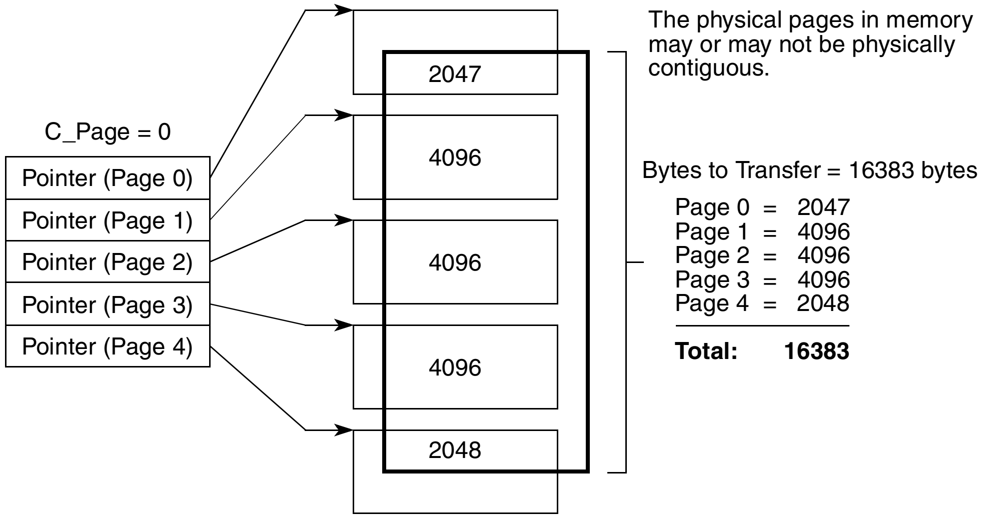


**Figure 21-50. Example Mapping of qTD Buffer Pointers to Buffer Pages**

The buffer pointer list in the qTD is long enough to support a maximum transfer size of 20K bytes. This case occurs when all five buffer pointers are used and the first offset is zero. A qTD handles a 16Kbyte buffer with any starting buffer alignment.

The host controller uses the C_Page field as an index value to determine which buffer pointer in the list should be used to start the current transaction. The host controller uses a different buffer pointer for each physical page of the buffer. This is always true, even if the buffer is physically contiguous.

The host controller must detect when the current transaction spans a page boundary and automatically move to the next available buffer pointer in the page pointer list. The next available pointer is reached by incrementing C_Page and pulling the next page pointer from the list. Software must ensure there are sufficient buffer pointers to move the amount of data specified in the Bytes to Transfer field.

Figure 21-50 illustrates a nominal example of how System software would initialize the buffer pointers list and the C_Page field for a transfer size of 16383 bytes. C_Page is cleared. The upper 20-bits of Page 0 references the start of the physical page. Current Offset (the lower 12-bits of queue head Dword 7) holds the offset in the page for example, 2049 (for example, 4096-2047). The remaining page pointers are set to reference the beginning of each subsequent 4K page.

For the first transaction on the qTD (assuming a 512-byte transaction), the host controller uses the first buffer pointer (page 0 because C_Page is cleared) and concatenates the Current Offset field. The 512 bytes are moved during the transaction, the Current Offset and Total Bytes to Transfer are adjusted by 512 and written back to the queue head working area.

During the 4th transaction, the host controller needs 511 bytes in page 0 and one byte in page 1. The host controller will increment C_Page (to 1) and use the page 1 pointer to move the final byte of the transaction. After the 4th transaction, the active page pointer is the page 1 pointer and Current Offset has rolled to one, and both are written back to the overlay area. The transactions continue for the rest of the buffer, with the host controller automatically moving to the next page pointer (that is, C_Page) when necessary. There are three conditions for how the host controller handles C_Page.

- The current transaction does not span a page boundary. The value of C_Page is not adjusted by the host controller.
- The current transaction does span a page boundary. The host controller must detect the page cross condition and advance to the next buffer while streaming data to/from the USB.
- The current transaction completes on a page boundary (that is, the last byte moved for the current transaction is the last byte in the page for the current page pointer). The host controller must increment C_Page before writing back status for the transaction.

Note that the only valid adjustment the host controller may make to C_Page is to increment by one.

### 21.6.10.2 Adding Interrupt Queue Heads to the Periodic Schedule

The link path(s) from the periodic frame list to a queue head establishes in which frames a transaction can be executed for the queue head. Queue heads are linked into the periodic schedule so they are polled at the appropriate rate. System software sets a bit in a queue head's S-Mask to indicate which micro-frame within a 1 millisecond period a transaction should be executed for the queue head. Software must ensure that all queue heads in the periodic schedule have S-Mask set to a non-zero value. An S-mask with a zero value in the context of the periodic schedule yields undefined results.

If the desired poll rate is greater than one frame, system software can use a combination of queue head linking and S-Mask values to spread interrupts of equal poll rates through the schedule so that the periodic bandwidth is allocated and managed in the most efficient manner possible. Some examples are illustrated in Table 21-65.

**Table 21-65. Example Periodic Reference Patterns for Interrupt Transfers**

| Frame #<br>Reference<br>Sequence | Description |
|---|---|
| 0, 2, 4, 6, 8, ....<br>S-Mask = 0x01 | A queue head for the bInterval of 2 milliseconds (16 micro-frames) is linked into the periodic schedule so that it is reachable from the periodic frame list locations indicated in the previous column. In addition, the S-Mask field in the queue head is set to 0x01, indicating that the transaction for the endpoint should be executed on the bus during micro-frame 0 of the frame. |
| 0, 2, 4, 6, 8, ...<br>S-Mask = 0x02 | Another example of a queue head with a bInterval of 2 milliseconds is linked into the periodic frame list at exactly the same interval as the previous example. However, the S-Mask is set to 0x02 indicating that the transaction for the endpoint should be executed on the bus during micro-frame 1 of the frame. |

### 21.6.10.3 Managing Transfer Complete Interrupts from Queue Heads

The host controller sets an interrupt to be signaled at the next interrupt threshold when the completed transfer (qTD) has an Interrupt on Complete (IOC) bit set, or whenever a transfer (qTD) completes with a short packet. If system software needs multiple qTDs to complete a client request (that is, like a control transfer) the intermediate qTDs do not require interrupts. System software may only need a single interrupt to notify it that the complete buffer has been transferred. System software may set IOC's to occur more frequently. A motivation for this may be that it wants early notification so that interface data structures can be re-used in a timely manner.

### 21.6.11 Ping Control

USB 2.0 defines an addition to the protocol for high-speed devices called Ping. Ping is required for all USB 2.0 High-speed bulk and control endpoints. Ping is not allowed for a split-transaction stream. This extension to the protocol eliminates the bad side-effects of Naking OUT endpoints. The Status field has a Ping State bit, which the host controller uses to determine the next actual PID it will use in the next transaction to the endpoint (see Table 21-53). The Ping State bit is only managed by the host controller for queue heads that meet all of the following criteria:

- The queue head is not an interrupt
- The EPS field equals High-Speed
- The PIDCode field equals OUT

Table 21-66 illustrates the state transition table for the host controller's responsibility for maintaining the PING protocol. Refer to Chapter 8 in the *USB Specification, Revision 2.0* for detailed description on the Ping protocol.

**Table 21-66. Ping Control State Transition Table**

| | Event | | |
|---|---|---|---|
| **Current** | **Host** | **Device** | **Next** |
| Do Ping | PING | Nak | Do Ping |
| Do Ping | PING | Ack | Do OUT |
| Do Ping | PING | XactErr[1] | Do Ping |
| Do Ping | PING | Stall | N/C[2] |
| Do OUT | OUT | Nak | Do Ping |
| Do OUT | OUT | Nyet | Do Ping[3] |
| Do OUT | OUT | Ack | Do OUT |
| Do OUT | OUT | XactErr[1] | Do Ping |
| Do OUT | OUT | Stall | N/C[2] |

[1] Transaction Error (XactErr) is any time the host misses the handshake.

[2] No transition change required for the Ping State bit. The Stall handshake results in the endpoint being halted (for example, Active cleared and Halt set). Software intervention is required to restart queue.

[3] A Nyet response to an OUT means that the device has accepted the data, but cannot receive any more at this time. Host must advance the transfer state and additionally, transition the Ping State bit to Do Ping.

The Ping State bit is described in Table 21-53. The defined ping protocol allows the host to be imprecise on the initialization of the ping protocol (that is, start in Do OUT when we don't know whether there is space on the device or not). The host controller manages the Ping State bit. System software sets the initial value in the queue head when it initializes a queue head. The host controller preserves the Ping State bit across all queue advancements. This means that when a new qTD is written into the queue head overlay area, the previous value of the Ping State bit is preserved.

## 21.6.12 Split Transactions

USB 2.0 defines extensions to the bus protocol for managing USB 1.x data streams through USB 2.0 hubs. This section describes how the host controller uses the interface data structures to manage data streams with full- and low-speed devices, connected below a USB 2.0 hub, utilizing the split transaction protocol. Refer to the USB 2.0 Specification for the complete definition of the split transaction protocol. Full- and low-speed devices are enumerated identically as high-speed devices, but the transactions to the full- and low-speed endpoints use the split-transaction protocol on the high-speed bus. The split transaction protocol is an encapsulation of (or wrapper around) the full- or low-speed transaction. The high-speed wrapper portion of the protocol is addressed to the USB 2.0 hub and transaction translator below which the full- or low-speed device is attached.

EHCI uses dedicated data structures for managing full-speed isochronous data streams. Control, Bulk and Interrupt are managed using the queuing data structures. The interface data structures need to be programmed with the device address and the transaction translator number of the USB 2.0 hub operating as the low-/full-speed host controller for this link. The following sections describe the details of how the host controller processes and manages the split transaction protocol.

### 21.6.12.1 Split Transactions for Asynchronous Transfers

A queue head in the asynchronous schedule with an EPS field indicating a full-or low-speed device indicates to the host controller that it must use split transactions to stream data for this queue head. All full-speed bulk and full-, low-speed control are managed via queue heads in the asynchronous schedule.

Software must initialize the queue head with the appropriate device address and port number for the transaction translator that is serving as the full-/low-speed host controller for the links connecting the endpoint. Software must also initialize the split transaction state bit (SplitXState) to Do-Start-Split. Finally, if the endpoint is a control endpoint, then system software must set the Control Transfer Type (C) bit in the queue head to a one. If this is not a control transfer type endpoint, the C bit must be initialized by software to be a zero. This information is used by the host controller to properly set the Endpoint Type (ET) field in the split transaction bus token. When the C bit is a zero, the split transaction token's ET field is set to indicate a bulk endpoint. When the C bit is a one, the split transaction token's ET field is set to indicate a control endpoint. Refer to Chapter 8 of *USB Specification, Revision 2.0* for details.

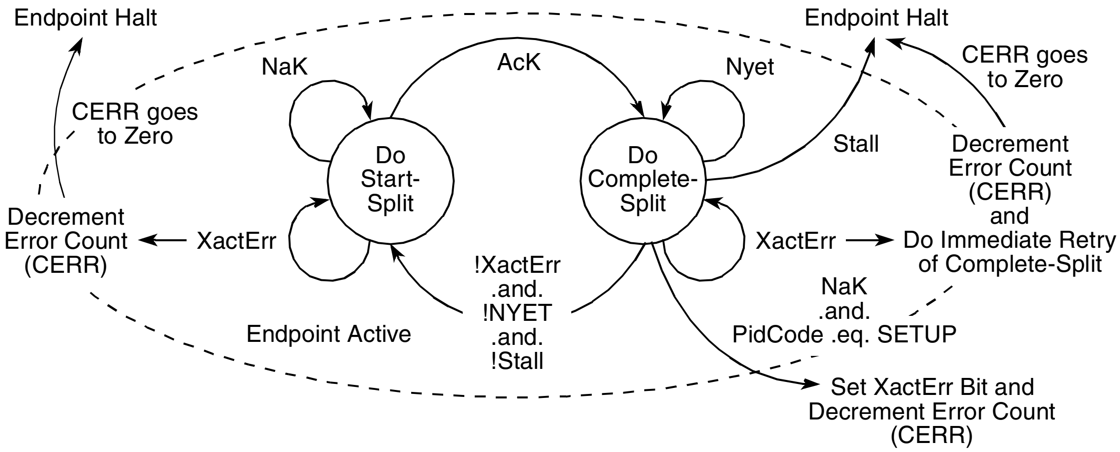

Endpoint Halt
Endpoint Halt
NaK
AcK
Nyet
CERR goes to Zero
CERR goes to Zero
Do Start-Split
Do Complete-Split
Stall
Decrement Error Count (CERR) and Do Immediate Retry of Complete-Split
Decrement Error Count (CERR)
XactErr
XactErr
!XactErr .and. !NYET .and. !Stall
Endpoint Active
NaK .and. PidCode .eq. SETUP
Set XactErr Bit and Decrement Error Count (CERR)

**Figure 21-51. Host Controller Asynchronous Schedule Split-Transaction State Machine**

#### 21.6.12.1.1 Asynchronous—Do-Start-Split

Do-Start-Split is the state which software must initialize a full- or low-speed asynchronous queue head. This state is entered from the Do-Complete-Split state only after a complete-split transaction receives a valid response from the transaction translator that is not a Nyet handshake.

For queue heads in this state, the host controller executes a start-split transaction to the transaction translator. If the bus transaction completes without an error and PID Code indicates an IN or OUT transaction, then the host controller reloads the error counter (Cerr). If it is a successful bus transaction and the PID Code indicates a SETUP, the host controller will not reload the error counter. If the transaction translator responds with a Nak, the queue head is left in this state, and the host controller proceeds to the next queue head in the asynchronous schedule.

If the host controller times out the transaction (no response, or bad response) the host controller decrements Cerr and proceeds to the next queue head in the asynchronous schedule.

#### 21.6.12.1.2 Asynchronous—Do-Complete-Split

This state is entered from the Do-Start-Split state only after a start-split transaction receives an Ack handshake from the transaction translator.

For queue heads in this state, the host controller executes a complete-split transaction to the transaction translator. If the transaction translator responds with a Nyet handshake, the queue head is left in this state, the error counter is reset and the host controller proceeds to the next queue head in the asynchronous schedule. When a Nyet handshake is received for a bus transaction where the queue head's PID Code indicates an IN or OUT, the host controller reloads the error counter (Cerr). When a Nyet handshake is received for a complete-split bus transaction where the queue head's PID Code indicates a SETUP, the host controller must not adjust the value of Cerr.

Independent of PID Code, the following responses have the indicated effects:

- Transaction Error (XactErr). Timeout/data CRC failure. The error counter (Cerr) is decremented by one and the complete split transaction is immediately retried (if possible). If there is not enough time in the micro-frame to execute the retry, the host controller ensures that the next time the host controller begins executing from the Asynchronous schedule, it must begin executing from this queue head. If another start-split (for some other endpoint) is sent to the transaction translator before the complete-split is really completed, the transaction translator could dump the results (which were never delivered to the host). This is why the core specification states the retries must be immediate. When the host controller returns to the asynchronous schedule in the next micro-frame, the first transaction from the schedule will be the retry for this endpoint. If Cerr went to zero, the host controller halts the queue.
- NAK. The target endpoint Nak’d the full- or low-speed transaction. The state of the transfer is not advanced and the state is exited. If the PID Code is a SETUP, then the Nak response is a protocol error. The XactErr status bit is set and the Cerr field is decremented.
- STALL. The target endpoint responded with a STALL handshake. The host controller sets the halt bit in the status byte, retires the qTD but does not attempt to advance the queue.

If the PID Code indicates an IN, then any of following responses are expected:

- DATA0/1. On reception of data, the host controller ensures the PID matches the expected data toggle and checks CRC. If the packet is good, the host controller advances the state of the transfer (for example, moves the data pointer by the number of bytes received, decrements the BytesToTransfer field by the number of bytes received, and toggles the dt bit). The host controller then exits this state. The response and advancement of transfer may trigger other processing events, such as retirement of the qTD and advancement of the queue.

  If the data sequence PID does not match the expected, the data is ignored, the transfer state is not advanced and this state is exited.

If the PID Code indicates an OUT/SETUP, then any of following responses are expected:

- ACK. The target endpoint accepted the data, so the host controller must advance the state of the transfer. The Current Offset field is incremented by Maximum Packet Length or Bytes to Transfer, whichever is less. The Bytes To Transfer field is decremented by the same amount and the data toggle bit (dt) is toggled. The host controller then exits this state.

  Advancing the transfer state may cause other processing events such as retirement of the qTD and advancement of the queue.

### 21.6.12.2 Split Transaction Interrupt

Split-transaction Interrupt-IN/OUT endpoints are managed using the same data structures used for high-speed interrupt endpoints. They both co-exist in the periodic schedule. Queue heads/qTDs offer the set of features required for reliable data delivery, which is characteristic to interrupt transfer types. The split-transaction protocol is managed completely within this defined functional transfer framework. For example, for a high-speed endpoint, the host controller will visit a queue head, execute a high-speed transaction (if criteria are met) and advance the transfer state (or not) depending on the results of the entire transaction. For low- and full-speed endpoints, the details of the execution phase are different (that is, takes more than one bus transaction to complete), but the remainder of the operational framework is intact.

#### 21.6.12.2.1 Split Transaction Scheduling Mechanisms for Interrupt

Full- and low-speed Interrupt queue heads have an EPS field indicating full- or low-speed and have a non-zero S-mask field. The host controller can detect this combination of parameters and assume the endpoint is a periodic endpoint. Low- and full-speed interrupt queue heads require the use of the split transaction protocol. The host controller sets the Endpoint Type (ET) field in the split token to indicate the transaction is an interrupt. These transactions are managed through a transaction translator's periodic pipeline. Software should not set these fields to indicate the queue head is an interrupt unless the queue head is used in the periodic schedule.

System software manages the per/transaction translator periodic pipeline by budgeting and scheduling exactly during which micro-frames the start-splits and complete-splits for each endpoint will occur. The characteristics of the transaction translator are such that the high-speed transaction protocol must execute during explicit micro-frames, or the data or response information in the pipeline is lost. Figure 21-52 illustrates the general scheduling boundary conditions that are supported by the EHCI periodic schedule and queue head data structure. The S and $C_n$ labels indicate micro-frames where software can schedule start-splits and complete splits (respectively).

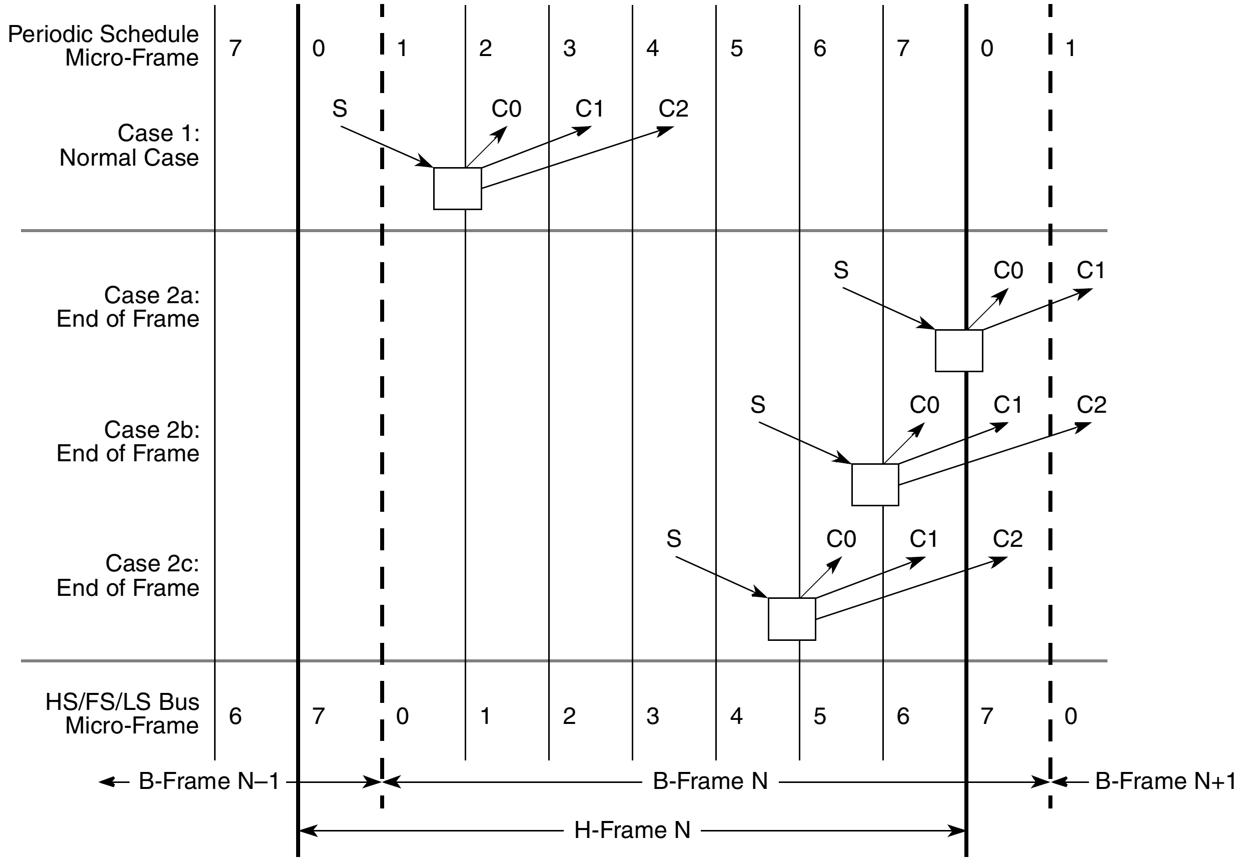


**Figure 21-52. Split Transaction, Interrupt Scheduling Boundary Conditions**

The scheduling cases are:

- Case 1: The normal scheduling case is where the entire split transaction is completely bounded by a frame (H-Frame in this case).
- Case 2a through Case 2c: The USB 2.0 hub pipeline rules states clearly, when and how many complete-splits must be scheduled to account for earliest to latest execution on the full/low-speed link. The complete-splits may span the H-Frame boundary when the start-split is in micro-frame 4 or later. When this occurs, the H-Frame to B-Frame alignment requires that the queue head be reachable from consecutive periodic frame list locations. System software cannot build an efficient schedule that satisfies this requirement unless it uses FSTNs. Figure 21-53 illustrates the general layout of the periodic schedule.

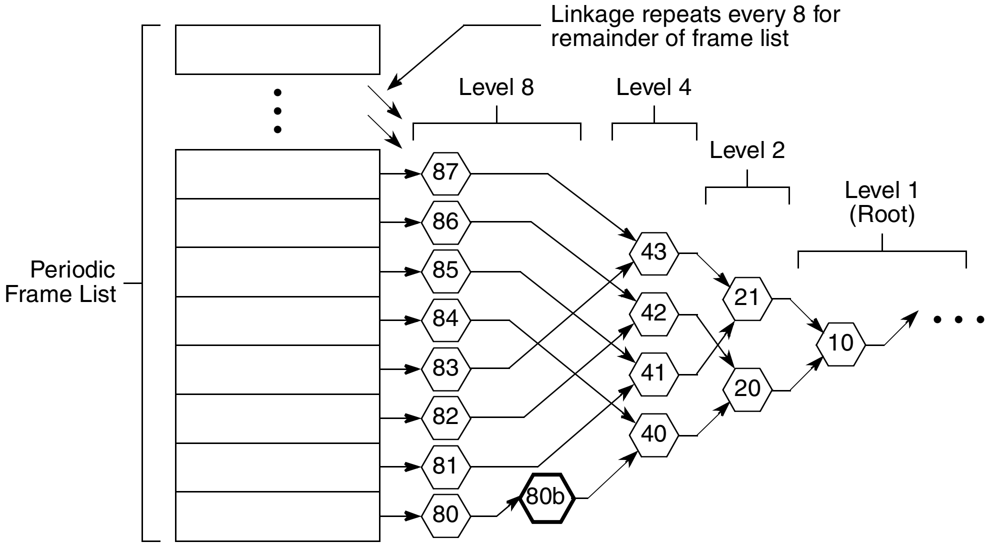


**Figure 21-53. General Structure of EHCI Periodic Schedule Utilizing Interrupt Spreading**

The periodic frame list is effectively the leaf level a binary tree, which is always traversed leaf to root. Each level in the tree corresponds to a $2^N$ poll rate. Software can efficiently manage periodic bandwidth on the USB by spreading interrupt queue heads that have the same poll rate requirement across all the available paths from the frame list. For example, system software can schedule eight poll rate 8 queue heads and account for them once in the high-speed bus bandwidth allocation.

When an endpoint is allocated an execution footprint that spans a frame boundary, the queue head for the endpoint must be reachable from consecutive locations in the frame list. An example would be if $8_{0b}$ were such an endpoint. Without additional support on the interface, to get $8_{0b}$ reachable at the correct time, software would have to link $8_1$ to $8_{0b}$. It would then have to move $4_1$ and everything linked after into the same path as $4_0$. This upsets the integrity of the binary tree and disallows the use of the spreading technique.

FSTN data structures are used to preserve the integrity of the binary-tree structure and enable the use of the spreading technique. Section 21.5.7, “Periodic Frame Span Traversal Node (FSTN),” defines the hardware and software operational model requirements for using FSTNs.

The following queue head fields are initialized by system software to instruct the host controller when to execute portions of the split-transaction protocol.

- SplitXState. This is a single bit residing in the Status field of a queue head (Table 21-53). This bit is used to track the current state of the split transaction.
- Frame S-mask. This is a bit-field where-in system software sets a bit corresponding to the micro-frame (within an H-Frame) that the host controller should execute a start-split transaction. This is always qualified by the value of the SplitXState bit in the Status field of the queue head. For example, referring to Figure 21-52, case one, the S-mask would have a value of 0b0000_0001 indicating that if the queue head is traversed by the host controller, and the SplitXState indicates Do_Start, and the current micro-frame as indicated by FRINDEX[2–0] is 0, then execute a start-split transaction.

- Frame C-mask. This is a bit-field where system software sets one or more bits corresponding to the micro-frames (within an H-Frame) that the host controller should execute complete-split transactions. The interpretation of this field is always qualified by the value of the SplitXState bit in the Status field of the queue head. For example, referring to Figure 21-52, case one, the C-mask would have a value of 0b0001_1100 indicating that if the queue head is traversed by the host controller, and the SplitXState indicates Do_Complete, and the current micro-frame as indicated by FRINDEX[2–0] is 2, 3, or 4, then execute a complete-split transaction. It is software's responsibility to ensure that the translation between H-Frames and B-Frames is correctly performed when setting bits in S-mask and C-mask.

### 21.6.12.2.2 Host Controller Operational Model for FSTNs

The FSTN data structure is used to manage Low/Full-speed interrupt queue heads that need to be reached from consecutive frame list locations (that is, boundary cases 2a through 2c). An FSTN is essentially a back pointer, similar in intent to the back pointer field in the siTD data structure.

This feature provides software a simple primitive to save a schedule position, redirect the host controller to traverse the necessary queue heads in the previous frame, then restore the original schedule position and complete normal traversal.

There are four components to the use of FSTNs:

- FSTN data structure, defined in Section 21.5.7, “Periodic Frame Span Traversal Node (FSTN).”
- A Save Place indicator; this is always an FSTN with its Back Path Link Pointer[T] bit cleared.
- A Restore indicator; this is always an FSTN with its Back Path Link Pointer[T] bit set.
- Host controller FSTN traversal rules.

When the host controller encounters an FSTN during micro-frames 2 through 7 it simply follows the node's Normal Path Link Pointer to access the next schedule data structure. Note that the FSTN's Normal Path Link Pointer[T] bit may set, which the host controller must interpret as the end of periodic list mark.

When the host controller encounters a Save-Place FSTN in micro-frames 0 or 1, it saves the value of the Normal Path Link Pointer and sets an internal flag indicating that it is executing in Recovery Path mode. Recovery Path mode modifies the host controller's rules for how it traverses the schedule and limits which data structures are considered for execution of bus transactions. The host controller continues executing in Recovery Path mode until it encounters a Restore FSTN or it determines that it has reached the end of the micro-frame.

The rules for schedule traversal and limited execution while in Recovery Path mode are:

- Always follow the Normal Path Link Pointer when it encounters an FSTN that is a Save-Place indicator. The host controller must not recursively follow Save-Place FSTNs. Therefore, while executing in Recovery Path mode, it must never follow an FSTN's Back Path Link Pointer.
- Do not process an siTD or iTD data structure; simply follow its Next Link Pointer.
- Do not process a QH (Queue Head) whose EPS field indicates a high-speed device; simply follow its Horizontal Link Pointer.
- When a QH's EPS field indicates a Full/Low-speed device, the host controller only considers it for execution if its SplitXState is DoComplete (note: this applies whether the PID Code indicates an

IN or an OUT). Refer to the *EHCI Specification* for a complete list of additional conditions that must be met in general for the host controller to issue a bus transaction. Note that the host controller must not execute a Start-split transaction while executing in Recovery Path mode. Refer to the *EHCI Specification* for special handling when in Recovery Path mode.

- Stop traversing the recovery path when it encounters an FSTN that is a Restore indicator. The host controller unconditionally uses the saved value of the Save-Place FSTN's Normal Path Link Pointer when returning to the normal path traversal. The host controller must clear the context of executing a Recovery Path when it restores schedule traversal to the Save-Place FSTN's Normal Path Link Pointer.

  If the host controller determines that there is not enough time left in the micro-frame to complete processing of the periodic schedule, it abandons traversal of the recovery path, and clears the context of executing a recovery path. The result is that at the start of the next consecutive micro-frame, the host controller starts traversal at the frame list.

An example traversal of a periodic schedule that includes FSTNs is illustrated in Figure 21-54.

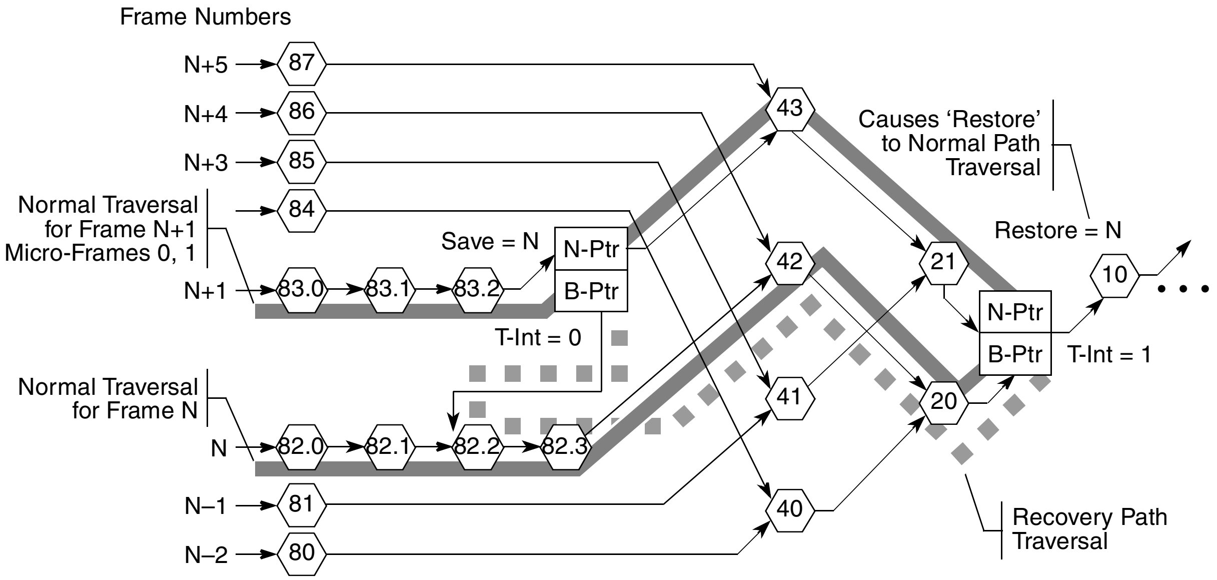


**Figure 21-54. Example Host Controller Traversal of Recovery Path via FSTNs**

In frame N (micro-frames 0-7), for this example, the host controller traverses all of the schedule data structures utilizing the Normal Path Link Pointers in any FSTNs it encounters. This is because the host controller has not yet encountered a Save-Place FSTN so it is not executing in Recovery Path mode. When it encounters the Restore FSTN, (Restore-N), during micro-frames 0 and 1, it uses Restore-N. Normal Path Link Pointer to traverse to the next data structure (that is, normal schedule traversal). This is because the host controller must use a Restore FSTN's Normal Path Link Pointer when not executing in a Recovery-Path mode. The nodes traversed during frame N include: {$8_{2.0}$, $8_{2.1}$, $8_{2.2}$, $8_{2.3}$, $4_2$, $2_0$, Restore-N, $1_0$ ...}.

In frame N+1 (micro-frames 0 and 1), when the host controller encounters Save-Path FSTN (Save-N), it observes that Save-N.Back Path Link Pointer.T-bit is zero (definition of a Save-Path indicator). The host controller saves the value of Save-N. Normal Path Link Pointer and follows Save-N.Back Path Link

Pointer. At the same time, it sets an internal flag indicating that it is now in Recovery Path mode (the recovery path is annotated in Figure 21-54 with a large dashed line). The host controller continues traversing data structures on the recovery path and executing only those bus transactions as noted above, on the recovery path until it reaches Restore FSTN (Restore-N). Restore-N.Back Path Link Pointer.T-bit is set (definition of a Restore indicator), so the host controller exits Recovery Path mode by clearing the internal Recovery Path mode flag and commences (restores) schedule traversal using the saved value of the Save-Place FSTN's Normal Path Link Pointer (for example, Save-N.Normal Path Link Pointer). The nodes traversed during these micro-frames include: {$8_{3.0}$, $8_{3.1}$, $8_{3.2}$, Save-A, $8_{2.2}$, $8_{2.3}$, $4_2$, $2_0$, Restore-N, $4_3$, $2_1$, Restore-N, 10 ...}.

In frame N+1 (micro-frames 2-7), when the host controller encounters Save-Path FSTN Save-N, it unconditionally follows Save-N.Normal Path Link Pointer. The nodes traversed during these micro-frames include: {$8_{3.0}$, $8_{3.1}$, $8_{3.2}$, Save-A, $4_3$, $2_1$, Restore-N, $1_0$ ...}.

### 21.6.12.2.3 Software Operational Model for FSTNs

Software must create a consistent, coherent schedule for the host controller to traverse. When using FSTNs, system software must adhere to the following rules:

- Each Save-Place indicator requires a matching Restore indicator.

  The Save-Place indicator is an FSTN with a valid Back Path Link Pointer and T-bit equal to zero. Note that Back Path Link Pointer[Typ] field must be set to indicate the referenced data structure is a queue head. The Restore indicator is an FSTN with its Back Path Link Pointer[T] bit set.

  A Restore FSTN may be matched to one or more Save-Place FSTNs. For example, if the schedule includes a poll-rate 1 level, then system software only needs to place a Restore FSTN at the beginning of this list in order to match all possible Save-Place FSTNs.
- If the schedule does not have elements linked at a poll-rate level of one, and one or more Save-Place FSTNs are used, then System Software must ensure the Restore FSTN's Normal Path Link Pointer's T-bit is set, as this will be use to mark the end of the periodic list.
- When the schedule does have elements linked at a poll rate level of one, a Restore FSTN must be the first data structure on the poll rate one list. All traversal paths from the frame list converge on the poll-rate one list. System software must ensure that Recovery Path mode is exited before the host controller is allowed to traverse the poll rate level one list.
- A Save-Place FSTN's Back Path Link Pointer must reference a queue head data structure. The referenced queue head must be reachable from the previous frame list location. In other words, if the Save-Place FSTN is reachable from frame list offset N, then the FSTN's Back Path Link Pointer must reference a queue head that is reachable from frame list offset N-1.

Software should make the schedule as efficient as possible. What this means in this context is that software should have no more than one Save-Place FSTN reachable in any single frame. Note there will be times when two (or more, depending on the implementation) could exist as full-/low-speed footprints change with bandwidth adjustments. This could occur, for example when a bandwidth rebalance causes system software to move the Save-Place FSTN from one poll rate level to another. During the transition, software must preserve the integrity of the previous schedule until the new schedule is in place.

#### 21.6.12.2.4 Tracking Split Transaction Progress for Interrupt Transfers

To correctly maintain the data stream, the host controller must be able to detect and report errors where data is lost. For interrupt-IN transfers, data is lost when it makes it into the USB 2.0 hub, but the USB 2.0 host system is unable to get it from the USB 2.0 hub and into the system before it expires from the transaction translator pipeline. When a lost data condition is detected, the queue is halted, thus signaling system software to recover from the error. A data-loss condition exists whenever a start-split is issued, accepted and successfully executed by the USB 2.0 hub, but the complete-splits get unrecoverable errors on the high-speed link, or the complete-splits do not occur at the correct times. One reason complete-splits might not occur at the right time would be due to host-induced system hold-offs that cause the host controller to miss bus transactions because it cannot get timely access to the schedule in system memory.

The same condition can occur for an interrupt-OUT, but the result is not an endpoint halt condition, but rather effects only the progress of the transfer. The queue head has the following fields to track the progress of each split transaction. These fields are used to keep incremental state about which (and when) portions have been executed.

- C-prog-mask. This is an eight-bit bit-vector where the host controller keeps track of which complete-splits have been executed. Due to the nature of the transaction translator periodic pipeline, the complete-splits need to be executed in-order. The host controller needs to detect when the complete-splits have not been executed in order. This can only occur due to system hold-offs where the host controller cannot get to the memory-based schedule. C-prog-mask is a simple bit-vector that the host controller sets one of the C-prog-mask bits for each complete-split executed. The bit position is determined by the micro-frame number in which the complete-split was executed. The host controller always checks C-prog-mask before executing a complete-split transaction. If the previous complete-splits have not been executed then it means one (or more) have been skipped and data has potentially been lost.
- FrameTag. This field is used by the host controller during the complete-split portion of the split transaction to tag the queue head with the frame number (H-Frame number) when the next complete split must be executed.
- S-bytes. This field can be used to store the number of data payload bytes sent during the start-split (if the transaction was an OUT). The S-bytes field must be used to accumulate the data payload bytes received during the complete-splits (for an IN).

#### 21.6.12.2.5 Split Transaction Execution State Machine for Interrupt

In the following section, all references to micro-frame are in the context of a micro-frame within an H-Frame.

As with asynchronous Full- and Low-speed endpoints, a split-transaction state machine is used to manage the split transaction sequence. Aside from the fields defined in the queue head for scheduling and tracking the split transaction, the host controller calculates one internal mechanism that is also used to manage the split transaction. The internal calculated mechanism is:

- cMicroFrameBit. This is a single-bit encoding of the current micro-frame number. It is an eight-bit value calculated by the host controller at the beginning of every micro-frame. It is calculated from the three least significant bits of the FRINDEX register (that is, cMicroFrameBit = (1 shifted-left(FRINDEX[2–0]))). The cMicroFrameBit has at most one bit asserted, which always corresponds to the current micro-frame number. For example, if the current micro-frame is 0, then cMicroFrameBit will equal 0b0000_0001.

The variable cMicroFrameBit is used to compare against the S-mask and C-mask fields to determine whether the queue head is marked for a start- or complete-split transaction for the current micro-frame.

Figure 21-55 illustrates how a complete interrupt split transaction is managed. There are two phases to each split transaction. The first is a single start-split transaction, which occurs when the SplitXState is at Do_Start and the single bit in cMicroFrameBit has a corresponding bit active in QH[S-mask]. The transaction translator does not acknowledge the receipt of the periodic start-split, so the host controller unconditionally transitions the state to Do_Complete. Due to the available jitter in the transaction translator pipeline, there will be more than one complete-split transaction scheduled by software for the Do_Complete state. This translates simply to the fact that there are multiple bits set in the QH[C-mask] field.

The host controller keeps the queue head in the Do_Complete state until the split transaction is complete (see definition below), or an error condition triggers the three-strikes-rule (for example, after the host tries the same transaction three times, and each encounters an error, the host controller stops retrying the bus transaction and halts the endpoint, thus requiring system software to detect the condition and perform system-dependent recovery).

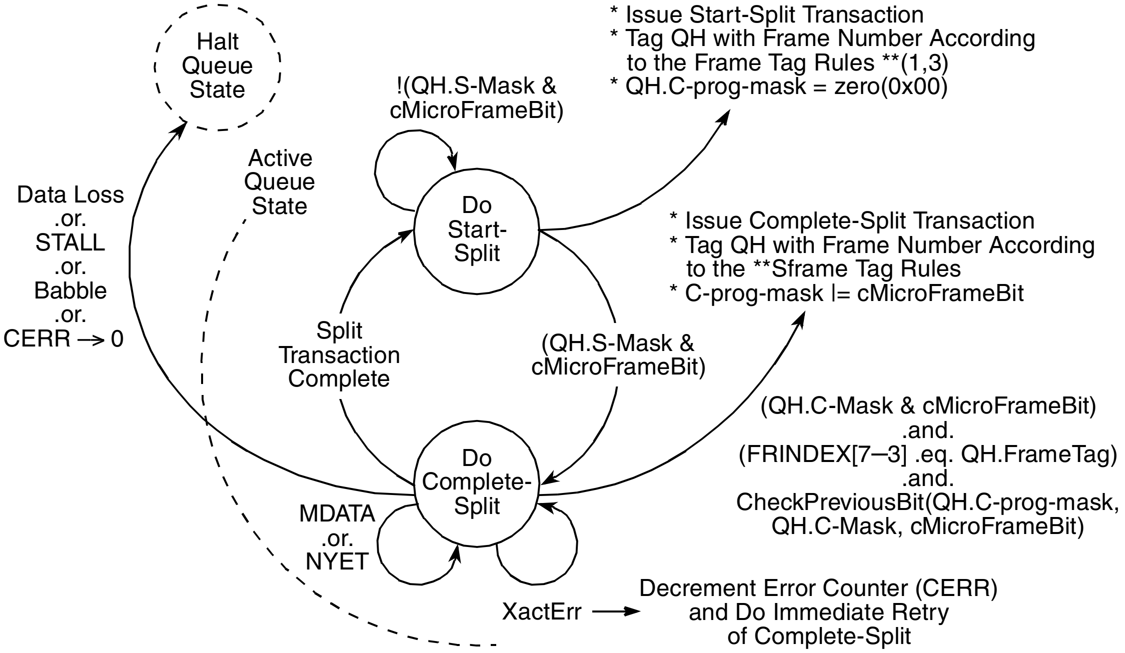

Halt Queue State
* Issue Start-Split Transaction
* Tag QH with Frame Number According to the Frame Tag Rules **(1,3)
* QH.C-prog-mask = zero(0x00)
!(QH.S-Mask & cMicroFrameBit)
Active Queue State
Data Loss .or. STALL .or. Babble .or. CERR → 0
Do Start-Split
* Issue Complete-Split Transaction
* Tag QH with Frame Number According to the **Sframe Tag Rules
* C-prog-mask |= cMicroFrameBit
Split Transaction Complete
(QH.S-Mask & cMicroFrameBit)
(QH.C-Mask & cMicroFrameBit) .and. (FRINDEX[7–3] .eq. QH.FrameTag) .and. CheckPreviousBit(QH.C-prog-mask, QH.C-Mask, cMicroFrameBit)
Do Complete-Split
MDATA .or. NYET
XactErr
Decrement Error Counter (CERR) and Do Immediate Retry of Complete-Split

Figure 21-55. Split Transaction State Machine for Interrupt

### 21.6.12.2.6 Periodic Interrupt—Do-Start-Split

This is the state software must initialize a full- or low-speed interrupt queue head StartXState bit. This state is entered from the Do_Complete Split state only after the split transaction is complete. This occurs when one of the following events occur: The transaction translator responds to a complete-split transaction with one of the following:

- NAK. A NAK response is a propagation of the full- or low-speed endpoint's NAK response.
- ACK. An ACK response is a propagation of the full- or low-speed endpoint's ACK response. Only occurs on an OUT endpoint.
- DATA 0/1. Only occurs for INs. Indicates that this is the last of the data from the endpoint for this split transaction.
- ERR. The transaction on the low-/full-speed link below the transaction translator had a failure (for example, timeout, bad CRC, etc.).
- NYET (and Last). The host controller issued the last complete-split and the transaction translator responded with a NYET handshake. This means that the start-split was not correctly received by the transaction translator, so it never executed a transaction to the full- or low-speed endpoint, see Section Periodic Interrupt - Do Complete Split for the definition of 'Last'.

Each time the host controller visits a queue head in this state (once within the Execute Transaction state), bit-wise ANDs QH[S-mask] with cMicroFrameBit to determine whether to execute a start-split. If the result is non-zero, then the host controller issues a start-split transaction. If the PID Code field indicates an IN transaction, the host controller must zero-out the QH[S-bytes] field. After the split-transaction has been executed, the host controller sets up state in the queue head to track the progress of the complete-split phase of the split transaction. Specifically, it records the expected frame number into QH[FrameTag] field, sets C-prog-mask to zero (0x00), and exits this state. Note that the host controller must not adjust the value of Cerr as a result of completion of a start-split transaction.

### 21.6.12.2.7 Periodic Interrupt—Do-Complete-Split

This state is entered unconditionally from the Do Start Split state after a start-split transaction is executed on the bus. Each time the host controller visits a queue head in this state (once within the Execute Transaction state), it checks to determine whether a complete-split transaction should be executed now.

There are four tests to determine whether a complete-split transaction should be executed.

- Test A. cMicroFrameBit is bit-wise ANDed with QH[C-mask] field. A non-zero result indicates that software scheduled a complete-split for this endpoint, during this micro-frame.
- Test B. QH[FrameTag] is compared with the current contents of FRINDEX[7–3]. An equal indicates a match.
- Test C. The complete-split progress bit vector is checked to determine whether the previous bit is set, indicating that the previous complete-split was appropriately executed. An example algorithm for this test is provided below:

```
Algorithm Boolean CheckPreviousBit(QH.C-prog-mask, QH.C-mask, cMicroFrameBit)
Begin
-- Return values:
-- TRUE - no error
-- FALSE - error
--
Boolean rvalue = TRUE;
previousBit = cMicroframeBit logical-rotate-right(1)
-- Bit-wise anding previousBit with C-mask indicates
-- whether there was an intent
-- to send a complete split in the previous micro-frame. So,
-- if the
-- 'previous bit' is set in C-mask, check C-prog-mask to
-- make sure it
-- happened.
If (previousBit bitAND QH.C-mask)then
                              If not(previousBit bitAND QH.C-prog-mask) then
        rvalue = FALSE;
    End if
End If
-- If the C-prog-mask already has a one in this bit position,
-- then an aliasing
-- error has occurred. It will probably get caught by the
-- FrameTag Test, but
-- at any rate it is an error condition that as detectable here
-- should not allow
-- a transaction to be executed.
If (cMicroFrameBit bitAND QH.C-prog-mask) then
rvalue = FALSE;
End if
return (rvalue)
End Algorithm
```

- Test D. Check to see if a start-split should be executed in this micro-frame. Note this is the same test performed in the Do Start Split state. Whenever it evaluates to TRUE and the controller is NOT processing in the context of a Recovery Path mode, it means a start-split should occur in this micro-frame. Test D and Test A evaluating to TRUE at the same time is a system software error. Behavior is undefined.

If (A .and. B .and. C .and. not(D)) then the host controller will execute a complete-split transaction. When the host controller commits to executing the complete-split transaction, it updates QH[C-prog-mask] by bit-ORing with cMicroFrameBit. On completion of the complete-split transaction, the host controller records the result of the transaction in the queue head and sets QH[FrameTag] to the expected H-Frame number. The effect to the state of the queue head and thus the state of the transfer depends on the response by the transaction translator to the complete-split transaction. The following responses have the effects (note that any responses that result in decrementing of the Cerr will result in the queue head being halted by the host controller if the result of the decrement is zero):

- NYET (and Last). On each NYET response, the host controller checks to determine whether this is the last complete-split for this split transaction. Last is defined in this context as the condition where all of the scheduled complete-splits have been executed. If it is the last complete-split (with a NYET response), then the transfer state of the queue head is not advanced (never received any data) and this state exited. The transaction translator must have responded to all the complete-splits with NYETs, meaning that the start-split issued by the host controller was not received. The start-split should be retried at the next poll period.
- The test for whether this is the Last complete split can be performed by XOR QH[C-mask] with QH[C-prog-mask]. If the result is all zeros then all complete-splits have been executed. When this condition occurs, the XactErr status bit is set and the Cerr field is decremented.
- NYET (and not Last). See above description for testing for Last. The complete-split transaction received a NYET response from the transaction translator. Do not update any transfer state (except for C-prog-mask and FrameTag) and stay in this state. The host controller must not adjust Cerr on this response.

- Transaction Error (XactErr). Timeout, data CRC failure, etc. The Cerr field is decremented and the XactErr bit in the Status field is set. The complete split transaction is immediately retried (if Cerr is non-zero).If there is not enough time in the micro-frame to complete the retry and the endpoint is an IN, or Cerr is decremented to a zero from a one, the queue is halted. If there is not enough time in the micro-frame to complete the retry and the endpoint is an OUT and Cerr is not zero, then this state is exited (that is, return to Do Start Split). This results in a retry of the entire OUT split transaction, at the next poll period. Refer to Chapter 11 Hubs (specifically the section on full- and low-speed interrupts) in the *USB Specification Revision 2.0* for detailed requirements on why these errors must be immediately retried.
- ACK. This can only occur if the target endpoint is an OUT. The target endpoint ACK'd the data and this response is a propagation of the endpoint ACK up to the host controller. The host controller must advance the state of the transfer. The Current Offset field is incremented by Maximum Packet Length or Bytes to Transfer, whichever is less. The field Bytes To Transfer is decremented by the same amount. And the data toggle bit (dt) is toggled. The host controller will then exit this state for this queue head. The host controller must reload Cerr with maximum value on this response. Advancing the transfer state may cause other process events such as retirement of the qTD and advancement of the queue.
- MDATA. This response will only occur for an IN endpoint. The transaction translator responded with zero or more bytes of data and an MDATA PID. The incremental number of bytes received is accumulated in QH[S-bytes]. The host controller must not adjust Cerr on this response.
- DATA0/1. This response may only occur for an IN endpoint. The number of bytes received is added to the accumulated byte count in QH[S-bytes]. The state of the transfer is advanced by the result and the host controller exits this state for this queue head.
- Advancing the transfer state may cause other processing events such as retirement of the qTD and advancement of the queue.
- If the data sequence PID does not match the expected, the entirety of the data received in this split transaction is ignored, the transfer state is not advanced and this state is exited.
- NAK. The target endpoint Nak'd the full- or low-speed transaction. The state of the transfer is not advanced, and this state is exited. The host controller must reload Cerr with maximum value on this response.
- ERR. There was an error during the full- or low-speed transaction. The ERR status bit is set, Cerr is decremented, the state of the transfer is not advanced, and this state is exited.
- STALL. The queue is halted (an exit condition of the Execute Transaction state). The status field bits: Active bit is cleared and the Halted bit is set and the qTD is retired. Responses which are not enumerated in the list or which are received out of sequence are illegal and may result in undefined host controller behavior. The other possible combinations of tests A, B, C, and D may indicate that data or response was lost. Table 21-67 lists the possible combinations and the appropriate action.

Table 21-67. Interrupt IN/OUT Do Complete Split State Execution Criteria

| Condition | Action | Description |
|---|---|---|
| not(A)<br>not(D) | Ignore QHD | Neither a start nor complete-split is scheduled for the current micro-frame.Host controller should continue walking the schedule. |
| A<br>not(C) | If PIDCode = IN<br>Halt QHDIf PIDCode = OUT<br>Retry start-split | Progress bit check failed. These means a complete-split has been missed. There is the possibility of lost data. If PID Code is an IN, then the Queue head must be halted.If PID Code is an OUT, then the transfer state is not advanced and the state exited (for example, start-split is retried). This is a host-induced error and does not effect Cerr. In either case, set the Missed Micro-frame bit in the status field to a one. |
| A<br>not(B)<br>C | If PIDCode = IN<br>Halt QHD<br>If PIDCode = OUT<br>Retry start-split | QH.FrameTag test failed. This means that exactly one or more H-Frames have been skipped. This means complete-splits and have missed. There is the possibility of lost data. If PID Code is an IN, then the Queue head must be halted.If PID Code is an OUT, then the transfer state is not advanced and the state exited (for example, start-split is retried). This is a host-induced error and does not effect Cerr. In either case, set the Missed Micro-frame bit in the status field to a one. |
| A<br>B<br>C<br>not(D) | Execute complete-split | This is the non-error case where the host controller executes a complete-split transaction. |
| D | If PIDCode = IN<br>Halt QHDIf PIDCode = OUT<br>Retry start-split | This is a degenerate case where the start-split was issued, but all of the complete-splits were skipped and all possible intervening opportunities to detect the missed data failed to fire. If PID Code is an IN, then the Queue head must be halted. If PID Code is an OUT, then the transfer state is not advanced and the state exited (for example, start-split is retried). This is a host-induced error and does not effect Cerr. In either case, set the Missed Micro-frame bit in the status field to a one. Note that when executing in the context of a Recovery Path mode, the host controller is allowed to process the queue head and take the actions indicated above, or it may wait until the queue head is visited in the normal processing mode. Regardless, the host controller must not execute a start-split in the context of a executing in a Recovery Path mode. |

### 21.6.12.2.8 Managing the QH[FrameTag] Field

The QH[FrameTag] field in a queue head is completely managed by the host controller. The rules for setting QH[FrameTag] are simple:

- Rule 1: If transitioning from Do Start Split to Do Complete Split and the current value of FRINDEX[2–0] is 6, QH[FrameTag] is set to FRINDEX[7–3] + 1. This accommodates split transactions whose start-split and complete-splits are in different H-Frames (case 2a, see Figure 21-52).
- Rule 2: If the current value of FRINDEX[2–0] is 7, QH[FrameTag] is set to FRINDEX[7–3] + 1. This accommodates staying in Do Complete Split for cases 2a, 2b, and 2c in Figure 21-52.
- Rule 3: If transitioning from Do_Start Split to Do Complete Split and the current value of FRINDEX[2–0] is not 6, or currently in Do Complete Split and the current value of (FRINDEX[2–0]) is not 7, FrameTag is set to FRINDEX[7–3]. This accommodates all other cases in Figure 21-52.

#### 21.6.12.2.9 Rebalancing the Periodic Schedule

System software must occasionally adjust a periodic queue head's S-mask and C-mask fields during operation. This need occurs when adjustments to the periodic schedule create a new bandwidth budget and one or more queue head's are assigned new execution footprints (that is, new S-mask and C-mask values).

It is imperative that system software must not update these masks to new values in the midst of a split transaction. In order to avoid any race conditions with the update, the host controller provides a simple assist to system software. System software sets the Inactivate-on-next-Transaction (I) bit to signal the host controller that it intends to update the S-mask and C-mask on this queue head. System software then waits for the host controller to observe the I-bit is set and transitions the Active bit to a zero. The rules for how and when the host controller clears the Active bit are:

- If the Active bit is cleared, no action is taken. The host controller does not attempt to advance the queue when the I-bit is set.
- If the Active bit is set and the SplitXState is DoStart (regardless of the value of S-mask), the host controller simply clears the Active bit. The host controller is not required to write the transfer state back to the current qTD. Note that if the S-mask indicates that a start-split is scheduled for the current micro-frame, the host controller must not issue the start-split bus transaction; it must clear the Active bit.

System software must save transfer state before setting the I-bit. This is required so that it can correctly determine what transfer progress (if any) occurred after the I-bit was set and the host controller executed it's final bus-transaction and cleared the Active bit.

After system software has updated the S-mask and C-mask, it must then reactivate the queue head. Since the Active bit and the I-bit cannot be updated with the same write, system software needs to use the following algorithm to coherently re-activate a queue head that has been stopped using the I-bit.

1. Set the Halted bit, then
2. Clear the I-bit, then
3. Set the Active bit and clear the Halted bit in the same write.

Setting the Halted bit inhibits the host controller from attempting to advance the queue between the time the I-bit is cleared and the Active bit is set.

### 21.6.12.3 Split Transaction Isochronous

Full-speed isochronous transfers are managed using the split-transaction protocol through a USB 2.0 transaction translator in a USB 2.0 hub. The host controller utilizes siTD data structure to support the special requirements of isochronous split-transactions. This data structure uses the scheduling model of isochronous TDs (see Section 21.6.8, “Managing Isochronous Transfers Using iTDs,” for the operational model of iTDs) with the contiguous data feature provided by queue heads. This simple arrangement allows a single isochronous scheduling model and adds the additional feature that all data received from the endpoint (per split transaction) must land into a contiguous buffer.

#### 21.6.12.3.1 Split Transaction Scheduling Mechanisms for Isochronous

Full-speed isochronous transactions are managed through a transaction translator's periodic pipeline. As with full- and low-speed interrupt, system software manages each transaction translator's periodic pipeline by budgeting and scheduling exactly during which micro-frames the start-splits and complete-splits for each full-speed isochronous endpoint occur. The requirements described in Section Split Transaction Scheduling Mechanisms for Interrupt apply. Figure 21-56 illustrates the general scheduling boundary conditions that are supported by the EHCI periodic schedule. The $S_n$ and $C_n$ labels indicate micro-frames where software can schedule start- and complete-splits (respectively). The H-Frame boundaries are marked with a large, solid bold vertical line. The B-Frame boundaries are marked with a large, bold, dashed line. The bottom of Figure 21-56 illustrates the relationship of an siTD to the H-Frame.

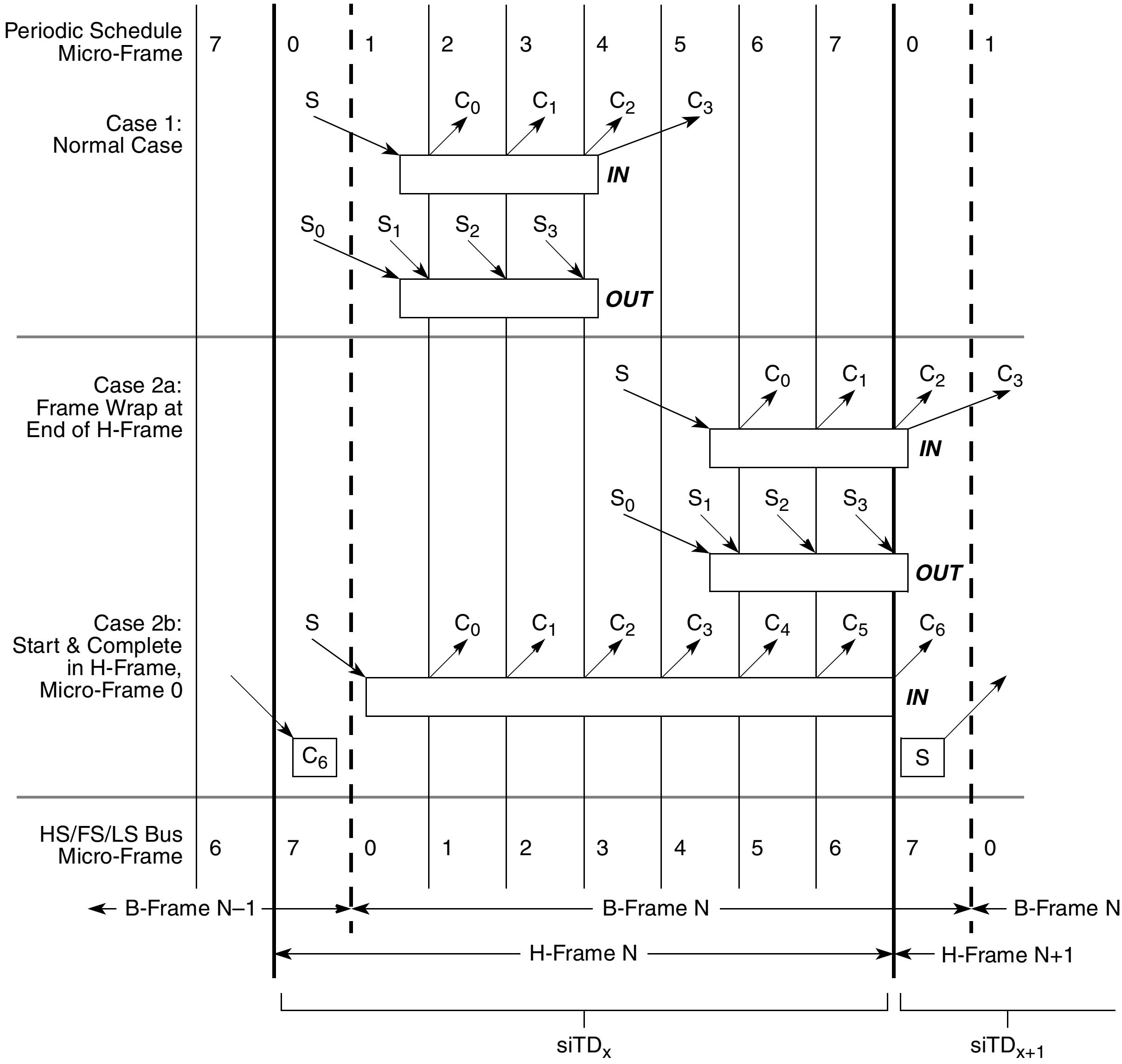


Figure 21-56. Split Transaction, Isochronous Scheduling Boundary Conditions

When the endpoint is an isochronous OUT, there are only start-splits, and no complete-splits. When the endpoint is an isochronous IN, there is at most one start-split and one to N complete-splits. The scheduling boundary cases are:

- Case 1: The entire split transaction is completely bounded by an H-Frame. For example, the start-splits and complete-splits are all scheduled to occur in the same H-Frame.
- Case 2a: This boundary case is where one or more (at most two) complete-splits of a split transaction IN are scheduled across an H-Frame boundary. This can only occur when the split transaction has the possibility of moving data in B-Frame, micro-frames 6 or 7 (H-Frame micro-frame 7 or 0). When an H-Frame boundary wrap condition occurs, the scheduling of the split transaction spans more than one location in the periodic list.(for example, it takes two siTDs in adjacent periodic frame list locations to fully describe the scheduling for the split transaction).

  Although the scheduling of the split transaction may take two data structures, all of the complete-splits for each full-speed IN isochronous transaction must use only one data pointer. For this reason, siTDs contain a back pointer.

  Software must never schedule full-speed isochronous OUTs across an H-Frame boundary.
- Case 2b: This case can only occur for a very large isochronous IN. It is the only allowed scenario where a start-split and complete-split for the same endpoint can occur in the same micro-frame. Software must enforce this rule by scheduling the large transaction first. Large is defined to be anything larger than 579 byte maximum packet size.

A subset of the same mechanisms employed by full- and low-speed interrupt queue heads are employed in siTDs to schedule and track the portions of isochronous split transactions. The following fields are initialized by system software to instruct the host controller when to execute portions of the split transaction protocol:

- SplitXState. This is a single bit residing in the Status field of an siTD (see Table 21-47). This bit is used to track the current state of the split transaction. The rules for managing this bit are described in Section 21.6.12.3.3, “Split Transaction Execution State Machine for Isochronous.”
- Frame S-mask. This is a bit-field wherein system software sets a bit corresponding to the micro-frame (within an H-Frame) that the host controller should execute a start-split transaction. This is always qualified by the value of the SplitXState bit. For example, referring to the IN example in Figure 21-56, case 1, the S-mask would have a value of 0b0000_0001 indicating that if the siTD is traversed by the host controller, and the SplitXState indicates Do Start Split, and the current micro-frame as indicated by FRINDEX[2–0] is 0, then execute a start-split transaction.
- Frame C-mask. This is a bit-field where system software sets one or more bits corresponding to the micro-frames (within an H-Frame) that the host controller should execute complete-split transactions. The interpretation of this field is always qualified by the value of the SplitXState bit. For example, referring to the IN example in Figure 21-56, case 1, the C-mask would have a value of 0b 0011_1100 indicating that if the siTD is traversed by the host controller, and the SplitXState indicates Do Complete Split, and the current micro-frame as indicated by FRINDEX[2–0] is 2, 3, 4, or 5, then execute a complete-split transaction.
- Back Pointer. This field in a siTD is used to complete an IN split-transaction using the previous H-Frame's siTD. This is only used when the scheduling of the complete-splits span an H-Frame boundary.

There exists a one-to-one relationship between a high-speed isochronous split transaction (including all start- and complete-splits) and one full-speed isochronous transaction. An siTD contains (amongst other things) buffer state and split transaction scheduling information. An siTD's buffer state always maps to one full-speed isochronous data payload. This means that for any full-speed transaction payload, a single siTD's data buffer must be used. This rule applies to both IN an OUTs. An siTD's scheduling information usually also maps to one high-speed isochronous split transaction. The exception to this rule is the H-Frame boundary wrap cases mentioned above.

The siTD data structure describes at most, one frame's worth of high-speed transactions and that description is strictly bounded within a frame boundary. Figure 21-57 illustrates some examples. On the top are examples of the full-speed transaction footprints for the boundary scheduling cases described above. In the middle are time-frame references for both the B-Frames (HS/FS/LS Bus) and the H-Frames. On the bottom is illustrated the relationship between the scope of an siTD description and the time references. Each H-Frame corresponds to a single location in the periodic frame list. The implication is that each siTD is reachable from a single periodic frame list location at a time.

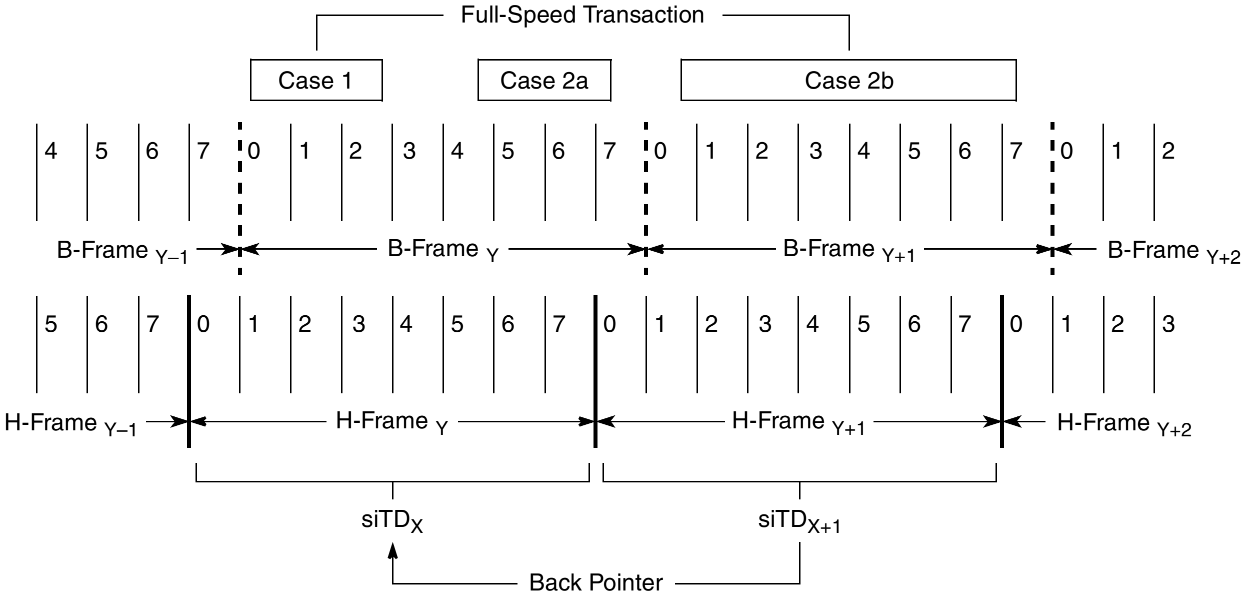


**Figure 21-57. siTD Scheduling Boundary Examples**

Each case is described below:

- Case 1: One siTD is sufficient to describe and complete the isochronous split transaction because the whole isochronous split transaction is tightly contained within a single H-Frame.
- Case 2a, 2b: Although both INs and OUTs can have these footprints, OUTs always take only one siTD to schedule. However, INs (for these boundary cases) require two siTDs to complete the scheduling of the isochronous split transaction. siTDX is used to always issue the start-split and the first N complete-splits. The full-speed transaction (for these cases) can deliver data on the full-speed bus segment during micro-frame 7 of H-Frame$_{Y+1}$, or micro-frame 0 of H-Frame$_{Y+2}$. The complete splits are scheduled using siTD$_{X+2}$ (not shown). The complete-splits to extract this data must use the buffer pointer from siTD$_{X+1}$. The only way for the host controller to reach siTD$_{X+1}$ from H-Frame$_{Y+2}$ is to use siTD$_{X+2}$'s back pointer.

Software must apply the following rules when calculating the schedule and linking the schedule data structures into the periodic schedule:

- Software must ensure that an isochronous split-transaction is started so that it will complete before the end of the B-Frame.
- Software must ensure that for a single full-speed isochronous endpoint, there is never a start-split and complete-split in H-Frame, micro-frame 1. This is mandated as a rule so that case 2a and case 2b can be discriminated. According to the core USB specification, the long isochronous transaction illustrated in Case 2b, could be scheduled so that the start-split was in micro-frame 1 of H-Frame N and the last complete-split would need to occur in micro-frame 1 of H-Frame N+1. However, it is impossible to discriminate between cases 2a and case 2b, which has significant impact on the complexity of the host controller.

#### 21.6.12.3.2 Tracking Split Transaction Progress for Isochronous Transfers

Isochronous endpoints do not employ the concept of a halt on error, however the host controller does identify and report per-packet errors observed in the data stream. This includes schedule traversal problems (skipped micro-frames), timeouts and corrupted data received.

In similar kind to interrupt split-transactions, the portions of the split transaction protocol must execute in the micro-frames they are scheduled. The queue head data structure used to manage full- and low-speed interrupt has several mechanisms for tracking when portions of a transaction have occurred. Isochronous transfers use siTDs for their transfers and the data structures are only reachable using the schedule in the exact micro-frame in which they are required (so all the mechanism employed for tracking in queue heads is not required for siTDs). Software has the option of reusing siTD several times in the complete periodic schedule. However, it must ensure that the results of split transaction N are consumed and the siTD re-initialized (activated) before the host controller gets back to the siTD (in a future micro-frame).

Split-transaction isochronous OUTs utilize a low-level protocol to indicate which portions of the split transaction data have arrived. Control over the low-level protocol is exposed in an siTD using the fields Transaction Position (TP) and Transaction Count (T-count). If the entire data payload for the OUT split transaction is larger than 188 bytes, there will be more than one start-split transaction, each of which require proper annotation. If host hold-offs occur, then the sequence of annotations received from the host will not be complete, which is detected and handled by the transaction translator. See Section 21.6.12.3.1, “Split Transaction Scheduling Mechanisms for Isochronous,” for a description on how these fields are used during a sequence of start-split transactions.

The fields siTD[T-Count] and siTD[TP] are used by the host controller to drive and sequence the transaction position annotations. It is the responsibility of system software to properly initialize these fields in each siTD. Once the budget for a split-transaction isochronous endpoint is established, S-mask, T-Count, and TP initialization values for all the siTD associated with the endpoint are constant. They remain constant until the budget for the endpoint is recalculated by software and the periodic schedule adjusted.

For IN-endpoints, the transaction translator simply annotates the response data packets with enough information to allow the host controller to identify the last data. As with split transaction Interrupt, it is the host controller's responsibility to detect when it has missed an opportunity to execute a complete-split. The following field in the siTD is used to track and detect errors in the execution of a split transaction for an IN isochronous endpoint.

- C-prog-mask. This is an eight-bit bit-vector where the host controller keeps track of which complete-splits have been executed. Due to the nature of the transaction translator periodic pipeline, the complete-splits need to be executed in-order. The host controller needs to detect when the complete-splits have not been executed in order. This can only occur due to system hold-offs where the host controller cannot get to the memory-based schedule. C-prog-mask is a simple bit-vector that the host controller sets a bit for each complete-split executed. The bit position is determined by the micro-frame (FRINDEX[2–0]) number in which the complete-split was executed. The host controller always checks C-prog-mask before executing a complete-split transaction. If the previous complete-splits have not been executed, then it means one (or more) have been skipped and data has potentially been lost. System software is required to initialize this field to zero before setting an siTD's Active bit to a one.

If a transaction translator returns with the final data before all of the complete-splits have been executed, the state of the transfer is advanced so that the remaining complete-splits are not executed. It is important to note that an IN siTD is retired based solely on the responses from the transaction translator to the complete-split transactions. This means, for example, that it is possible for a transaction translator to respond to a complete-split with an MDATA PID. The number of bytes in the MDATA's data payload could cause the siTD[Total Bytes to Transfer] field to decrement to zero. This response can occur, before all of the scheduled complete-splits have been executed. In other interface, data structures (for example, high-speed data streams through queue heads), the transition of Total Bytes to Transfer to zero signals the end of the transfer and results in clearing the Active bit. However, in this case, the result has not been delivered by the transaction translator and the host must continue with the next complete-split transaction to extract the residual transaction state. This scenario occurs because of the pipeline rules for a transaction translator. In summary, the periodic pipeline rules require that on a micro-frame boundary, the transaction translator holds the final two bytes received (if it has not seen an End Of Packet (EOP)) in the full-speed bus pipe stage and gives the remaining bytes to the high-speed pipeline stage. At the micro-frame boundary, the transaction translator could have received the entire packet (including both CRC bytes) but not received the packet EOP. In the next micro-frame, the transaction translator responds with an MDATA and sends all of the data bytes (with the two CRC bytes being held in the full-speed pipeline stage). This could cause the siTD to decrement it's Total Bytes to Transfer field to zero, indicating it has received all expected data. The host must still execute one more (scheduled) complete-split transaction in order to extract the results of the full-speed transaction from the transaction translator (for example, the transaction translator may have detected a CRC failure, and this result must be forwarded to the host).

If the host experiences hold-offs that cause the host controller to skip one or more (but not all) scheduled split transactions for an isochronous OUT, then the protocol to the transaction translator is not consistent and the transaction translator detects and reacts to the problem. Likewise, for host hold-offs that cause the host controller to skip one or more (but not all) scheduled split transactions for an isochronous IN, the C-prog-mask is used by the host controller to detect errors. However, if the host experiences a hold-off that causes it to skip all of an siTD, or an siTD expires during a host hold off (for example, a hold-off occurs and the siTD is no longer reachable by the host controller in order for it to report the hold-off event), then system software must detect that the siTDs have not been processed by the host controller (for example, state not advanced) and report the appropriate error to the client driver.

### 21.6.12.3.3 Split Transaction Execution State Machine for Isochronous

In this section, all references to micro-frame are in the context of a micro-frame within an H-Frame.

If the Active bit in the Status byte is a zero, the host controller ignores the siTD and continues traversing the periodic schedule. Otherwise the host controller processes the siTD as specified below. A split transaction state machine is used to manage the split-transaction protocol sequence. The host controller uses the fields defined in Section 21.6.12.3.2, “Tracking Split Transaction Progress for Isochronous Transfers,” plus the variable cMicroFrameBit defined in Section 21.6.12.2.5, “Split Transaction Execution State Machine for Interrupt,” to track the progress of an isochronous split transaction. Figure 21-58 illustrates the state machine for managing an siTD through an isochronous split transaction. Bold, dotted circles denote the state of the Active bit in the Status field of a siTD. The Bold, dotted arcs denote the transitions between these states. Solid circles denote the states of the split transaction state machine and the solid arcs denote the transitions between these states. Dotted arcs and boxes reference actions that take place either as a result of a transition or from being in a state.

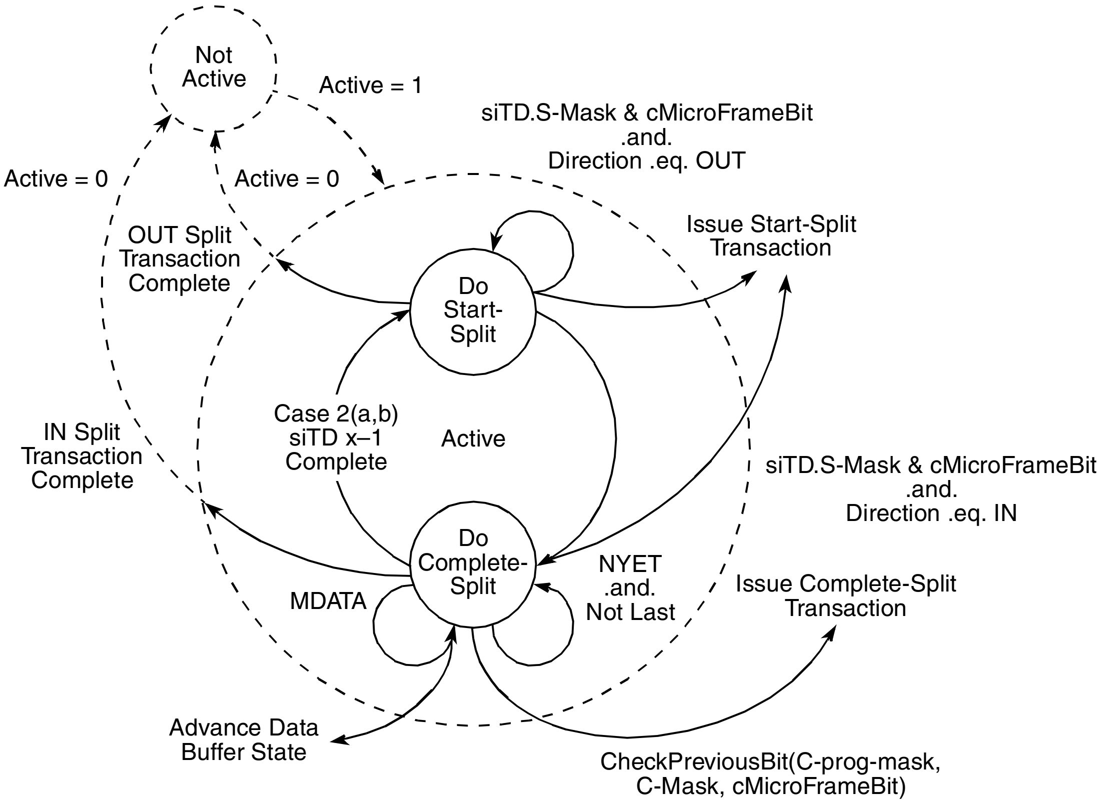

Not Active
Active = 1
siTD.S-Mask & cMicroFrameBit .and. Direction .eq. OUT
Active = 0
Active = 0
Issue Start-Split Transaction
OUT Split Transaction Complete
Do Start-Split
Case 2(a,b) siTD x–1 Complete
Active
IN Split Transaction Complete
siTD.S-Mask & cMicroFrameBit .and. Direction .eq. IN
Do Complete-Split
NYET .and. Not Last
MDATA
Issue Complete-Split Transaction
Advance Data Buffer State
CheckPreviousBit(C-prog-mask, C-Mask, cMicroFrameBit)

**Figure 21-58. Split Transaction State Machine for Isochronous**

### 21.6.12.3.4 Periodic Isochronous—Do-Start-Split

Isochronous split transaction OUTs use only this state. An siTD for a split-transaction isochronous IN is either initialized to this state, or the siTD transitions to this state from Do Complete Split when a case 2a (IN) or 2b scheduling boundary isochronous split-transaction completes.

Each time the host controller reaches an active siTD in this state, it checks the siTD[S-mask] against cMicroFrameBit. If there is a one in the appropriate position, the siTD executes a start-split transaction.

By definition, the host controller cannot reach an siTD at the wrong time. If the I/O field indicates an IN, then the start-split transaction includes only the extended token plus the full-speed token. Software must initialize the siTD[Total Bytes To Transfer] field to the number of bytes expected. This is usually the maximum packet size for the full-speed endpoint. The host controller exits this state when the start-split transaction is complete.

The remainder of this section is specific to an isochronous OUT endpoint (that is, the I/O field indicates an OUT). When the host controller executes a start-split transaction for an isochronous OUT it includes a data payload in the start-split transaction. The memory buffer address for the data payload is constructed by concatenating siTD[Current Offset] with the page pointer indicated by the page select field (siTD[P]). A zero in this field selects Page 0 and a 1 selects Page 1. During the start-split for an OUT, if the data transfer crosses a page boundary during the transaction, the host controller must detect the page cross, update the siTD[P] bit from a zero to a one, and begin using the siTD Page 1 with siTD[Current Offset] as the memory address pointer. The field siTD[TP] is used to annotate each start-split transaction with the indication of which part of the split-transaction data the current payload represents (ALL, BEGIN, MID, END). In all cases, the host controller simply uses the value in siTD[TP] to mark the start-split with the correct transaction position code.

T-count is always initialized to the number of start-splits for the current frame. TP is always initialized to the first required transaction position identifier. The scheduling boundary case (see Figure 21-57) is used to determine the initial value of TP. The initial cases are summarized in Table 21-68.

Table 21-68. Initial Conditions for OUT siTD TP and T-Count Fields

| Case | T-count | TP | Description |
|---|---|---|---|
| 1, 2a | =1 | ALL | When the OUT data payload is less than (or equal to) 188 bytes, only one start-split is required to move the data. The one start-split must be marked with an ALL. |
| 1, 2a | !=1 | BEGIN | When the OUT data payload is greater than 188 bytes more than one start-split must be used to move the data. The initial start-split must be marked with a BEGIN. |

After each start-split transaction is complete, the host controller updates T-count and TP appropriately so that the next start-split is correctly annotated. Table 21-69 illustrates all of the TP and T-count transitions, which must be accomplished by the host controller.

Table 21-69. Transaction Position (TP)/Transaction Count (T-Count) Transition Table

| TP | T-count next | TP next | Description |
|---|---|---|---|
| ALL | 0 | N/A | Transition from ALL, to done. |
| BEGIN | 1 | END | Transition from BEGIN to END. Occurs when T-count starts at 2. |
| BEGIN | !=1 | MID | Transition from BEGIN to MID. Occurs when T-count starts at greater than 2. |
| MID | !=1 | MID | TP stays at MID while T-count is not equal to 1 (for example, greater than 1). This case can occur for any of the scheduling boundary cases where the T-count starts greater than 3. |
| MID | 1 | END | Transition from MID to END. This case can occur for any of the scheduling boundary cases where the T-count starts greater than 2. |

The start-split transactions do not receive a handshake from the transaction translator, so the host controller always advances the transfer state in the siTD after the bus transaction is complete. To advance the transfer state the following operations take place:

- The siTD[Total Bytes To Transfer] and the siTD[Current Offset] fields are adjusted to reflect the number of bytes transferred.
- The siTD[P] (page select) bit is updated appropriately.
- The siTD[TP] and siTD[T-count] fields are updated appropriately as defined in Table 21-69.

These fields are then written back to the memory based siTD. The S-mask is fixed for the life of the current budget. As mentioned above, TP and T-count are set specifically in each siTD to reflect the data to be sent from this siTD. Therefore, regardless of the value of S-mask, the actual number of start-split transactions depends on T-count (or equivalently, Total Bytes to Transfer). The host controller must clear the Active bit when it detects that all of the schedule data has been sent to the bus. The preferred method is to detect when T-Count decrements to zero as a result of a start-split bus transaction. Equivalently, the host controller can detect when Total Bytes to Transfer decrements to zero. Either implementation must ensure that if the initial condition is Total Bytes to Transfer is equal to zero and T-count is equal to a one, then the host controller will issue a single start-split, with a zero-length data payload. Software must ensure that TP, T-count and Total Bytes to Transfer are set to deliver the appropriate number of bus transactions from each siTD. An inconsistent combination will yield undefined behavior.

If the host experiences hold-offs that cause the host controller to skip start-split transactions for an OUT transfer, the state of the transfer will not progress appropriately. The transaction translator observes protocol violations in the arrival of the start-splits for the OUT endpoint (that is, the transaction position annotation is incorrect as received by the transaction translator).

Example scenarios are described in Section 21.6.12.3.7, “Split Transaction for Isochronous—Processing Example.”

The host controller can optionally track the progress of an OUT split transaction by setting appropriate bits in the siTD[C-prog-mask] as it executes each scheduled start-split. The checkPreviousBit() algorithm defined in Section 21.6.12.3.5, “Periodic Isochronous—Do Complete Split,” can be used prior to executing each start-split to determine whether start-splits were skipped. The host controller can use this mechanism to detect missed micro-frames. It can then clear the siTD's Active bit and stop execution of this siTD. This saves on both memory and high-speed bus bandwidth.

#### 21.6.12.3.5 Periodic Isochronous—Do Complete Split

This state is only used by a split-transaction isochronous IN endpoint. This state is entered unconditionally from the Do Start State after a start-split transaction is executed for an IN endpoint. Each time the host controller visits an siTD in this state, it conducts a number of tests to determine whether it should execute a complete-split transaction. The individual tests are listed below. The sequence they are applied depends on which micro-frame the host controller is currently executing which means that the tests might not be applied until after the siTD referenced from the back pointer has been fetched.

- Test A. cMicroFrameBit is bit-wise ANDed with the siTD[C-mask] field. A non-zero result indicates that software scheduled a complete-split for this endpoint, during this micro-frame. This test is always applied to a newly fetched siTD that is in this state.

- Test B. The siTD[C-prog-mask] bit vector is checked to determine whether the previous complete splits have been executed. An example algorithm is given below (this is slightly different than the algorithm used in Section 21.6.12.2.7, “Periodic Interrupt—Do-Complete-Split”). The sequence in which this test is applied depends on the current value of FRINDEX[2–0]. If FRINDEX[2–0] is 0 or 1, it is not applied until the back pointer has been used. Otherwise it is applied immediately.

```
Algorithm Boolean CheckPreviousBit(siTD.C-prog-mask, siTD.C-mask, cMicroFrameBit)
Begin
     Boolean rvalue = TRUE;
     previousBit = cMicroFrameBit rotate-right(1)
     -- Bit-wise anding previousBit with C-mask indicates whether there
     -- was an intent to send a complete split in the previous micro-
     -- frame. So, if the 'previous bit' is set in C-mask, check
     -- C-prog-mask to make sure it happened.
     if previousBit bitAND siTD.C-mask then
          if not (previousBit bitAND siTD.C-prog-mask) then
                                                                   rvalue = FALSE
          End if
     End if
     Return rvalue
End Algorithm
```

If Test A is true and FRINDEX[2–0] is zero or one, then this is a case 2a or 2b scheduling boundary (see Figure 21-56). See Section 21.6.12.3.6, “Complete-Split for Scheduling Boundary Cases 2a, 2b,” for details in handling this condition.

If Test A and Test B evaluate to true, then the host controller executes a complete-split transaction using the transfer state of the current siTD. When the host controller commits to executing the complete-split transaction, it updates QH[C-prog-mask] by bit-ORing with cMicroFrameBit. The transfer state is advanced based on the completion status of the complete-split transaction. To advance the transfer state of an IN siTD, the host controller must:

- Decrement the number of bytes received from siTD[Total Bytes To Transfer]
- Adjust siTD[Current Offset] by the number of bytes received
- Adjust the siTD[P] (page select) field if the transfer caused the host controller to use the next page pointer
- Set any appropriate bits in the siTD[Status] field, depending on the results of the transaction.

Note that if the host controller encounters a condition where siTD[Total Bytes To Transfer] is zero, and it receives more data, the host controller must not write the additional data to memory. The siTD[Status-Active] bit must be cleared and the siTD[Status-Babble Detected] bit must be set. The fields siTD[Total Bytes To Transfer], siTD[Current Offset], and siTD[P] are not required to be updated as a result of this transaction attempt.

The host controller accepts (assuming good data packet CRC and sufficient room in the buffer as indicated by the value of siTD[Total Bytes To Transfer]) MDATA and DATA0/1 data payloads up to and including 192 bytes. The host controller may optionally clear siTD[Status-Active] and set siTD[Status-Babble

Detected] when it receives MDATA or DATA0/1 with a data payload of more than 192 bytes. The following responses have the noted effects:

- ERR. The full-speed transaction completed with a time-out or bad CRC and this is a reflection of that error to the host. The host controller sets the ERR bit in the siTD[Status] field and clears the Active bit.
- Transaction Error (XactErr). The complete-split transaction encounters a Timeout, CRC16 failure, etc. The siTD[Status] field XactErr field is set and the complete-split transaction must be retried immediately. The host controller must use an internal error counter to count the number of retries as a counter field is not provided in the siTD data structure. The host controller will not retry more than two times. If the host controller exhausts the retries or the end of the micro-frame occurs, the Active bit is cleared.
- DATAx (0 or 1). This response signals that the final data for the split transaction has arrived. The transfer state of the siTD is advanced and the Active bit is cleared. If the Bytes To Transfer field has not decremented to zero (including the reception of the data payload in the DATAx response), then less data than was expected, or allowed for was actually received. This short packet event does not set the USB interrupt status bit (USBSTS[UI]) to a one. The host controller will not detect this condition.
- NYET (and Last). On each NYET response, the host controller also checks to determine whether this is the last complete-split for this split transaction. Last was defined in Section Periodic Interrupt - Do Complete Split. If it is the last complete-split (with a NYET response), then the transfer state of the siTD is not advanced (never received any data) and the Active bit is cleared. No bits are set in the Status field because this is essentially a skipped transaction. The transaction translator must have responded to all the scheduled complete-splits with NYETs, meaning that the start-split issued by the host controller was not received. This result should be interpreted by system software as if the transaction was completely skipped. The test for whether this is the last complete split can be performed by XORing C-mask with C-prog-mask. A zero result indicates that all complete-splits have been executed.
- MDATA (and Last). See above description for testing for Last. This can only occur when there is an error condition. Either there has been a babble condition on the full-speed link, which delayed the completion of the full-speed transaction, or software set up the S-mask and/or C-masks incorrectly. The host controller must set the XactErr bit and clear the Active bit.
- NYET (and not Last). See above description for testing for Last. The complete-split transaction received a NYET response from the transaction translator. Do not update any transfer state (except for C-prog-mask) and stay in this state.
- MDATA (and not Last). The transaction translator responds with an MDATA when it has partial data for the split transaction. For example, the full-speed transaction data payload spans from micro-frame X to X+1 and during micro-frame X, the transaction translator responds with an MDATA and the data accumulated up to the end of micro-frame X. The host controller advances the transfer state to reflect the number of bytes received.

If Test A succeeds, but Test B fails, it means that one or more of the complete-splits have been skipped. The host controller sets the Missed Micro-Frame status bit and clears the Active bit.

#### 21.6.12.3.6 Complete-Split for Scheduling Boundary Cases 2a, 2b

Boundary cases 2a and 2b (INs only) (see Figure 21-56) require that the host controller use the transaction state context of the previous siTD to finish the split transaction. Table 21-70 enumerates the transaction state fields.

Table 21-70. Summary siTD Split Transaction State

| Buffer State | Status | Execution Progress |
|---|---|---|
| Total Bytes To Transfer<br>P (page select)<br>Current Offset<br>TP (transaction position)<br>T-count (transaction count) | All bits in the status field | C-prog-mask |

**NOTE**

TP and T-count are used only for Host to Device (OUT) endpoints.

If software has budgeted the schedule of this data stream with a frame wrap case, then it must initialize the siTD[Back Pointer] field to reference a valid siTD and have the T bit in the siTD[Back Pointer] field cleared. Otherwise, software must set the T bit in siTD[Back Pointer]. The host controller's rules for interpreting when to use the siTD[Back Pointer] field are listed below. These rules apply only when the siTD's Active bit is a one and the SplitXState is Do Complete Split.

- When cMicroFrameBit is a 0x1 and the $\text{siTD}_X$[Back Pointer] T-bit is zero, or
- If cMicroFrameBit is a 0x2 and siTDX[S-mask[0]] is zero

When either of these conditions apply, then the host controller must use the transaction state from $\text{siTD}_{X-1}$.

In order to access $\text{siTD}_{X-1}$, the host controller reads on-chip the siTD referenced from $\text{siTD}_X$[Back Pointer].

The host controller must save the entire state from $\text{siTD}_X$ while processing $\text{siTD}_{X-1}$. This is to accommodate for case 2b processing. The host controller must not recursively walk the list of siTD[Back Pointers].

If $\text{siTD}_{X-1}$ is active (Active bit is set and SplitXStat is Do Complete Split), then both Test A and Test B are applied as described above. If these criteria to execute a complete-split are met, the host controller executes the complete split and evaluates the results as described above. The transaction state (see Table 21-70) of $\text{siTD}_{X-1}$ is appropriately advanced based on the results and written back to memory. If the resultant state of $\text{siTD}_{X-1}$'s Active bit is a one, then the host controller returns to the context of $\text{siTD}_X$, and follows its next pointer to the next schedule item. No updates to $\text{siTD}_X$ are necessary.

If $\text{siTD}_{X-1}$ is active (Active bit is set and SplitXStat is Do Start Split), then the host controller must clear the Active bit and set the Missed Micro-Frame status bit and the resultant status is written back to memory.

If $\text{siTD}_{X-1}$'s Active bit is cleared, (because it was cleared when the host controller first visited $\text{siTD}_{X-1}$ via $\text{siTD}_X$'s back pointer, it transitioned to zero as a result of a detected error, or the results of $\text{siTD}_{X-1}$'s complete-split transaction cleared it), then the host controller returns to the context of $\text{siTD}_X$ and transitions its SplitXState to Do Start Split. The host controller then determines whether the case 2b start split boundary condition exists (that is, if cMicroframeBit is 1 and $\text{siTD}_X$[S-mask[0]] is 1). If this criterion is met the host controller immediately executes a start-split transaction and appropriately advances the transaction state of siTD$_X$, then follows siTD$_X$[Next Pointer] to the next schedule item. If the criterion is not met, the host controller simply follows siTD$_X$[Next Pointer] to the next schedule item. Note that in the case of a 2b boundary case, the split-transaction of siTD$_{X-1}$ will have its Active bit cleared when the host controller returns to the context of siTD$_X$. Also, note that software should not initialize an siTD with C-mask bits 0 and 1 set and an S-mask with bit 0 set. This scheduling combination is not supported and the behavior of the host controller is undefined.

### 21.6.12.3.7 Split Transaction for Isochronous—Processing Example

There is an important difference between how the hardware/software manages the isochronous split transaction state machine and how it manages the asynchronous and interrupt split transaction state machines. The asynchronous and interrupt split transaction state machines are encapsulated within a single queue head. The progress of the data stream depends on the progress of each split transaction. In some respects, the split-transaction state machine is sequenced using the Execute Transaction queue head traversal state machine.

Isochronous is a pure time-oriented transaction/data stream. The interface data structures are optimized to efficiently describe transactions that need to occur at specific times. The isochronous split-transaction state machine must be managed across these time-oriented data structures. This means that system software must correctly describe the scheduling of split-transactions across more than one data structure.

Then the host controller must make the appropriate state transitions at the appropriate times, in the correct data structures.

For example, Table 21-71 illustrates a few frames worth of scheduling required to schedule a case 2a full-speed isochronous data stream.

Table 21-71. Example Case 2a—Software Scheduling siTDs for an IN Endpoint

| siTDX | | Micro-Frames | | | | | | | | InitialSplitXState |
|---|---|---|---|---|---|---|---|---|---|---|
| # | Masks | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |
| X | S-Mask | | | | | 1 | | | | Do Start Split |
| | C-Mask | 1 | 1 | | | | | 1 | 1 | |
| X+1 | S-Mask | | | | | 1 | | | | Do Complete Split |
| | C-Mask | 1 | 1 | | | | | 1 | 1 | |
| X+2 | S-Mask | | | | | 1 | | | | Do Complete Split |
| | C-Mask | 1 | 1 | | | | | 1 | 1 | |
| X+3 | S-Mask | Repeats previous pattern | | | | | | | | Do Complete Split |
| | C-Mask | | | | | | | | | |

This example shows the first three siTDs for the transaction stream. Since this is the case-2a frame-wrap case, S-masks of all siTDs for this endpoint have a value of 0x10 (a one bit in micro-frame 4) and C-mask value of 0xC3 (one-bits in micro-frames 0,1, 6 and 7). Additionally, software ensures that the Back Pointer field of each siTD references the appropriate siTD data structure (and the Back Pointer T-bits are cleared).

The initial SplitXState of the first siTD is Do Start Split. The host controller will visit the first siTD eight times during frame X. The C-mask bits in micro-frames 0 and 1 are ignored because the state is Do Start Split. During micro-frame 4, the host controller determines that it can run a start-split (and does) and changes SplitXState to Do Complete Split. During micro-frames 6 and 7, the host controller executes complete-splits. Notice the siTD for frame X+1 has it's SplitXState initialized to Do Complete Split. As the host controller continues to traverse the schedule during H-Frame X+1, it will visit the second siTD eight times. During micro-frames 0 and 1 it will detect that it must execute complete-splits.

During H-Frame X+1, micro-frame 0, the host controller detects that siTD$_{X+1}$'s Back Pointer[T] bit is a zero, saves the state of siTD$_{X+1}$ and fetches siTD$_X$. It executes the complete split transaction using the transaction state of siTD$_X$. If the siTD$_X$ split transaction is complete, siTD's Active bit is cleared and results written back to siTD$_X$. The host controller retains the fact that siTD$_X$ is retired and transitions the SplitXState in siTD$_{X+1}$ to Do Start Split. At this point, the host controller is prepared to execute the start-split for siTD$_{X+1}$ when it reaches micro-frame 4. If the split-transaction completes early (transaction-complete is defined in Section 21.6.12.3.5, “Periodic Isochronous—Do Complete Split”), that is, before all the scheduled complete-splits have been executed, the host controller changes siTD$_X$[SplitXState] to Do Start Split early and naturally skips the remaining scheduled complete-split transactions. For this example, siTD$_{X+1}$ does not receive a DATA0 response until H-Frame X+2, micro-frame 1.

During H-Frame X+2, micro-frame 0, the host controller detects that siTD$_{X+2}$'s Back Pointer[T] bit is zero, saves the state of siTD$_{X+2}$ and fetches siTD$_{X+1}$. As described above, it executes another split transaction, receives an MDATA response, updates the transfer state, but does not modify the Active bit. The host controller returns to the context of siTD$_{X+2}$, and traverses it's next pointer without any state change updates to siTD$_{X+2}$.

During H-Frame X+2, micro-frame 1, the host controller detects siTD$_{X+2}$'s S-mask[0] bit is zero, saves the state of siTD$_{X+2}$ and fetches siTD$_{X+1}$. It executes another complete-split transaction, receives a DATA0 response, updates the transfer state and clears the Active bit. It returns to the state of siTD$_{X+2}$ and changes its SplitXState to Do Start Split. At this point, the host controller is prepared to execute start-splits for siTD$_{X+2}$ when it reaches micro-frame 4.

## 21.6.13 Port Test Modes

EHCI host controllers implement the port test modes Test J_State, Test K_State, Test_Packet, Test Force_Enable, and Test SE0_NAK as described in the *USB Specification Revision 2.0*. The required, port test sequence is (assuming the CF-bit in the CONFIGFLAG register is set):

- Disable the periodic and asynchronous schedules by clearing the USBCMD[ASE] and USBCMD[PSE].
- Place all enabled root ports into the suspended state by setting the Suspend bit in the PORTSC register (PORTSC[SUSP]).
- Clear USBCMD[RS] (run/stop) and wait for USBSTS[HCH] to transition to a one. Note that an EHCI host controller implementation may optionally allow port testing with RS set. However, all host controllers must support port testing with RS cleared and HCH set.

- Set the Port Test Control field in the port under test PORTSC register to the value corresponding to the desired test mode. If the selected test is Test_Force_Enable, then USBCMD[RS] must then be transitioned back to one, in order to enable transmission of SOFs out of the port under test.
- When the test is complete, system software must ensure the host controller is halted (HCH bit is a one) then it terminates and exits test mode by setting USBCMD[RST].

### 21.6.14 Interrupts

The EHCI host controller hardware provides interrupt capability based on a number of sources. There are several general groups of interrupt sources:

- Interrupts as a result of executing transactions from the schedule (success and error conditions),
- Host controller events (Port change events, etc.), and
- Host controller error events

All transaction-based sources are maskable through the host controller's Interrupt Enable register (USBINTR). Additionally, individual transfer descriptors can be marked to generate an interrupt on completion. This section describes each interrupt source and the processing that occurs in response to the interrupt.

During normal operation, interrupts may be immediate or deferred until the next interrupt threshold occurs. The interrupt threshold is a tunable parameter via the Interrupt Threshold Control field in the USBCMD register. The value of this register controls when the host controller generates an interrupt on behalf of normal transaction execution. When a transaction completes during an interrupt interval period, the interrupt signaling the completion of the transfer will not occur until the interrupt threshold occurs. For example, the default value is eight micro-frames. This means that the host controller will not generate interrupts any more frequently than once every eight micro-frames.

Section 21.6.14.2.4, “Host System Error” details effects of a host system error.

If an interrupt has been scheduled to be generated for the current interrupt threshold interval, the interrupt is not signaled until after the status for the last complete transaction in the interval has been written back to system memory. This may sometimes result in the interrupt not being signaled until the next interrupt threshold.

Initial interrupt processing is the same, regardless of the reason for the interrupt. When an interrupt is signaled by the hardware, CPU control is transferred to host controller's USB interrupt handler. The precise mechanism to accomplish the transfer is OS specific. For this discussion it is just assumed that control is received. When the interrupt handler receives control, its first action is to reads the USBSTS. It then acknowledges the interrupt by clearing all of the interrupt status bits by writing ones to these bit positions. The handler then determines whether the interrupt is due to schedule processing or some other event. After acknowledging the interrupt, the handler (via an OS-specific mechanism), schedules a deferred procedure call (DPC) which will execute later. The DPC routine processes the results of the schedule execution. The precise mechanisms used are beyond the scope of this document.

### NOTE

The only method software should use for acknowledging an interrupt is by transitioning the appropriate status bits in the USBSTS register from a one to a zero.

## 21.6.14.1 Transfer/Transaction Based Interrupts

These interrupt sources are associated with transfer and transaction progress. They are all dependent on the next interrupt threshold.

### 21.6.14.1.1 Transaction Error

A transaction error is any error that caused the host controller to think that the transfer did not complete successfully. Table 21-72 lists the events/responses that the host can observe as a result of a transaction. The effects of the error counter and interrupt status are summarized in the following paragraphs. Most of these errors set the XactErr status bit in the appropriate interface data structure.

**Table 21-72. Summary of Transaction Errors**

| Event/ Result | Queue Head/qTD/iTD/siTD Side Effects | | USBSTS[USBERRINT] |
|---|---|---|---|
| | Cerr | Status Field | |
| CRC | –1 | XactErr set | 1[1] |
| Timeout | –1 | XactErr set | 1[1] |
| Bad PID[2] | –1 | XactErr set | 1[1] |
| Babble | N/A | See Section 21.6.14.1.2, “Serial Bus Babble” | 1 |
| Buffer Error | N/A | See Section 21.6.14.1.3, “Data Buffer Error” | |

[1] If occurs in a queue head, then USBERRINT is asserted only when Cerr counts down from a one to a zero. In addition the queue is halted.

[2] The host controller received a response from the device, but it could not recognize the PID as a valid PID.

There is a small set of protocol errors that relate only when executing a queue head and fit under the umbrella of a WRONG PID error that are significant to explicitly identify. When these errors occur, the XactErr status bit in the queue head is set and the Cerr field is decremented. When the PID Code indicates a SETUP, the following responses are protocol errors and result in XactErr bit being set and the Cerr field being decremented.

- EPS field indicates a high-speed device and it returns a Nak handshake to a SETUP.
- EPS field indicates a high-speed device and it returns a Nyet handshake to a SETUP.
- EPS field indicates a low- or full-speed device and the complete-split receives a Nak handshake.

### 21.6.14.1.2 Serial Bus Babble

When a device transmits more data on the USB than the host controller is expecting for this transaction, it is defined to be babbling. In general, this is called a Packet Babble. When a device sends more data than the Maximum Length number of bytes, the host controller sets the Babble Detected bit to a one and halts the endpoint if it is using a queue head. Maximum Length is defined as the minimum of Total Bytes to Transfer and Maximum Packet Size. The Cerr field is not decremented for a packet babble condition (only applies to queue heads). A babble condition also exists if IN transaction is in progress at High-speed EOF2 point. This is called a frame babble. A frame babble condition is recorded into the appropriate schedule data structure. In addition, the host controller must disable the port to which the frame babble is detected.

USBSTS[UEI] (USB error interrupt) is set and if the USBINTR[UEE] (USB error interrupt enable) is set, then a hardware interrupt is signaled to the system at the next interrupt threshold. The host controller must never start an OUT transaction that babbles across a micro-frame EOF.

**NOTE**

When a host controller detects a data PID mismatch, it must either: disable the packet babble checking for the duration of the bus transaction or do packet babble checking based solely on Maximum Packet Size. The USB core specification defines the requirements on a data receiver when it receives a data PID mismatch (for example, expects a DATA0 and gets a DATA1 or visa-versa). In summary, it must ignore the received data and respond with an ACK handshake, in order to advance the transmitter's data sequence.The EHCI interface allows system software to provide buffers for a Control, Bulk or Interrupt IN endpoint that are not an even multiple of the maximum packet size specified by the device. Whenever a device misses an ACK for an IN endpoint, the host and device are out of synchronization with respect to the progress of the data transfer. The host controller may have advanced the transfer to a buffer that is less than maximum packet size. The device re-sends its maximum packet size data packet, with the original data PID, in response to the next IN token. In order to properly manage the bus protocol, the host controller must disable the packet babble check when it observes the data PID mismatch.

#### 21.6.14.1.3 Data Buffer Error

This event indicates that an overrun of incoming data or a underrun of outgoing data has occurred for this transaction. This would generally be caused by the host controller not being able to access required data buffers in memory within necessary latency requirements. These conditions are not considered transaction errors, and do not effect the error count in the queue head. When these errors do occur, the host controller records the fact the error occurred by setting the Data Buffer Error bit in the queue head, iTD or siTD.

If the data buffer error occurs on a non-isochronous IN, the host controller will not issue a handshake to the endpoint. This forces the endpoint to resend the same data (and data toggle) in response to the next IN to the endpoint.

If the data buffer error occurs on an OUT, the host controller must corrupt the end of the packet so that it cannot be interpreted by the device as a good data packet. Simply truncating the packet is not considered acceptable. An acceptable implementation option is to 1's complement the CRC bytes and send them. There are other options suggested in the transaction translator section of the *USB Specification Revision 2.0*.

#### 21.6.14.1.4 USB Interrupt (Interrupt on Completion (IOC))

Transfer Descriptors (iTDs, siTDs, and queue heads (qTDs)) contain a bit that can be set to cause an interrupt on their completion. The completion of the transfer associated with that schedule item causes USBSTS[UI] (USB interrupt) to be set. In addition, if a short packet is encountered on an IN transaction associated with a queue head, then this event also causes USBINT to be set. If USBINTR[UE] (USB interrupt enable) is set, a hardware interrupt is signaled to the system at the next interrupt threshold. If the completion is because of errors, USBSTS[UEI] (USB error interrupt) is also set.

#### 21.6.14.1.5 Short Packet

Reception of a data packet that is less than the endpoint's Max Packet size during Control, Bulk or Interrupt transfers signals the completion of the transfer. Whenever a short packet completion occurs during a queue head execution, USBSTS[UI] (USB interrupt bit) is set. If the USB interrupt enable bit is set (USBINTR[UE]), a hardware interrupt is signaled to the system at the next interrupt threshold.

### 21.6.14.2 Host Controller Event Interrupts

These interrupt sources are independent of the interrupt threshold (with the one exception being the Interrupt on Async Advance.

#### 21.6.14.2.1 Port Change Events

Port registers contain status and status change bits. When the status change bits are set, the host controller sets the USBSTS[PCI]. If the port change interrupt enable bit (PCE) in the USBINTR register is set, the host controller issues a hardware interrupt. The port status change bits in PORTSC include:

- Connect change status (CSC)
- Port enable/disable change (PEC)
- Over-current change (OCC)
- Force port resume (FPR)

#### 21.6.14.2.2 Frame List Rollover

This event indicates that the host controller has wrapped the frame list. The current programmed size of the frame list effects how often this interrupt occurs. If the frame list size is 1024, then the interrupt occurs every 1024 milliseconds, if it is 512, then it occurs every 512 milliseconds, etc. When a frame list rollover is detected, the host controller sets the frame list rollover bit, USBSTS[FRI]. If USBINTR[FRE] is set (frame list rollover enable), the host controller issues a hardware interrupt. This interrupt is not delayed to the next interrupt threshold.

#### 21.6.14.2.3 Interrupt on Async Advance

This event is used for deterministic removal of queue heads from the asynchronous schedule. Whenever the host controller advances the on-chip context of the asynchronous schedule, it evaluates the value of USBCMD[IAA]. If it is set, it sets USBSTS[AAI]. If USBINTR[AAE] is set, the host controller issues a hardware interrupt at the next interrupt threshold. A detailed explanation of this feature is described in Section 21.6.9.2, “Removing Queue Heads from Asynchronous Schedule.”

#### 21.6.14.2.4 Host System Error

The host controller is a bus master and any interaction between the host controller and the system may experience errors. The type of host error may be catastrophic to the host controller making it impossible for the host controller to continue in a coherent fashion. Behavior for these types of errors is to halt the host controller. Host-based error must result in the following actions:

- USBCMD[RS] is cleared.
- USBSTS[SEI] and USBSTS[HCH] register are set
- If the host system error enable bit, USBINTR[SEE] is set, the host controller issues a hardware interrupt. This interrupt is not delayed to the next interrupt threshold.

Table 21-73 summarizes the required actions taken on the various host errors.

**Table 21-73. Summary Behavior on Host System Errors**

| Cycle Type | Master Abort | Target Abort | Data Phase Parity |
|---|---|---|---|
| Frame list pointer fetch (read) | Fatal | Fatal | Fatal |
| siTD fetch (read) | Fatal | Fatal | Fatal |
| siTD status write-back (write) | Fatal | Fatal | Fatal |
| iTD fetch (read) | Fatal | Fatal | Fatal |
| iTD status write-back (write) | Fatal | Fatal | Fatal |
| qTD fetch (read) | Fatal | Fatal | Fatal |
| qHD status write-back (write) | Fatal | Fatal | Fatal |
| Data write | Fatal | Fatal | Fatal |
| Data read | Fatal | Fatal | Fatal |

**NOTE**

After a host system error, software must reset the host controller using USBCMD[RST] before re-initializing and restarting the host controller.

## 21.7 Device Data Structures

This section defines the interface data structures used to communicate control, status, and data between device controller driver (DCD) software and the device controller. The data structure definitions in this chapter support a 32-bit memory buffer address space. The interface consists of device queue heads and transfer descriptors.

**NOTE**

Software must ensure that no interface data structure reachable by the device controller spans a 4K-page boundary.

The data structures defined in the section are (from the device controller's perspective) a mix of read-only and read/ writable fields. The device controller must preserve the read-only fields on all data structure writes.

The USB module includes DCD software called the USB 2.0 Device API. The device API provides an easy to use Application Program Interface for developing device (peripheral) applications. The device API incorporates and abstracts for the application developer all of the elements of the program interface.

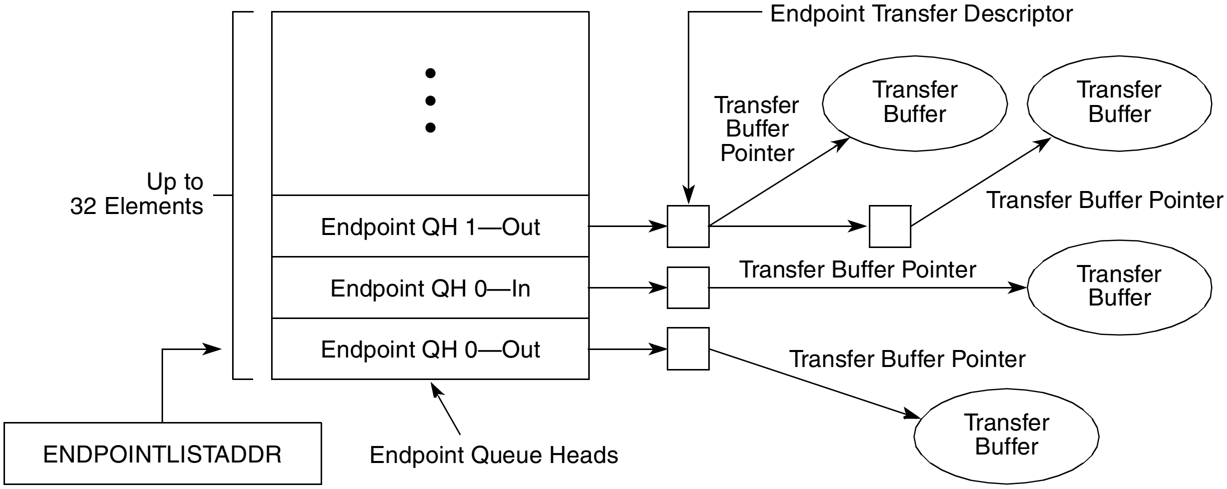


**Figure 21-59. End Point Queue Head Organization**

## 21.7.1 Endpoint Queue Head

The device Endpoint Queue Head (dQH) is where all transfers are managed. The dQH is a 48-byte data structure, but must be aligned on 64-byte boundaries. During priming of an endpoint, the dTD (device transfer descriptor) is copied into the overlay area of the dQH, which starts at the nextTD pointer DWord and continues through the end of the buffer pointers DWords. After a transfer is complete, the dTD status DWord is updated in the dTD pointed to by the currentTD pointer. While a packet is in progress, the overlay area of the dQH is used as a staging area for the dTD so that the Device Controller can access needed information with little minimal latency.

Figure 21-60 shows the Endpoint Queue Head structure.

<table>
<tr><th>31</th><th>30</th><th>29</th><th>28</th><th>27</th><th>26</th><th>25</th><th>24</th><th>23</th><th>22</th><th>21</th><th>20</th><th>19</th><th>18</th><th>17</th><th>16</th><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th><th>offset</th></tr>
<tr><td colspan="2">Mult</td><td>zlt</td><td colspan="2">00</td><td colspan="11">Maximum Packet Length</td><td>ios</td><td colspan="15">000_0000_0000_0000</td><td>0x00</td></tr>
<tr><td colspan="27">Current dTD Pointer[1]</td><td colspan="5">0_0000</td><td>0x04</td></tr>
<tr><td colspan="27">Next dTD Pointer[1]</td><td colspan="4">0000</td><td>T[1]</td><td>0x08[2]</td></tr>
<tr><td>0</td><td colspan="15">Total Bytes[1]</td><td>ioc[1]</td><td colspan="3">000</td><td colspan="2">MultO[1]</td><td colspan="2">00</td><td colspan="8">Status[1]</td><td>0x0C[2]</td></tr>
<tr><td colspan="20">Buffer Pointer (Page 0)[1]</td><td colspan="12">Current Offset[1]</td><td>0x10[2]</td></tr>
<tr><td colspan="20">Buffer Pointer (Page 1)[1]</td><td colspan="12">Reserved</td><td>0x14[2]</td></tr>
<tr><td colspan="20">Buffer Pointer (Page 2)[1]</td><td colspan="12">Reserved</td><td>0x18[2]</td></tr>
<tr><td colspan="20">Buffer Pointer (Page 3)[1]</td><td colspan="12">Reserved</td><td>0x1C[2]</td></tr>
<tr><td colspan="20">Buffer Pointer (Page 4)[1]</td><td colspan="12">Reserved</td><td>0x20[2]</td></tr>
<tr><td colspan="32">Reserved</td><td>0x24</td></tr>
<tr><td colspan="32">Set-up Buffer Bytes 3–0[1]</td><td>0x28</td></tr>
<tr><td colspan="32">Set-up Buffer Bytes 7–4[1]</td><td>0x2C</td></tr>
</table>

**Figure 21-60. Endpoint Queue Head Layout**

[1] Device controller read/write; all others read-only.

[2] Offsets 0x08 through 0x20 contain the transfer overlay.

### 21.7.1.1 Endpoint Capabilities/Characteristics

This DWord specifies static information about the endpoint, in other words, this information does not change over the lifetime of the endpoint. Device Controller software should not attempt to modify this information while the corresponding endpoint is enabled.

**Table 21-74. Endpoint Capabilities/Characteristics**

| Bits | Name | Description |
|---|---|---|
| 31–30 | Mult | Mult. This field is used to indicate the number of packets executed per transaction description as given by the following:<br>00 - Execute N Transactions as demonstrated by the USB variable length packet protocol where N is computed using the Maximum Packet Length (dQH) and the Total Bytes field (dTD)<br>01 Execute 1 Transaction.<br>10 Execute 2 Transactions.<br>11 Execute 3 Transactions.<br>**Note:** Non-ISO endpoints must set Mult = 00.<br>**Note:** ISO endpoints must set Mult = 01, 10, or 11 as needed. |
| 29 | zlt | Zero length termination select. This bit is used to indicate when a zero length packet is used to terminate transfers where to total transfer length is a multiple. This bit is not relevant for Isochronous transfers.<br>0 Enable zero length packet to terminate transfers equal to a multiple of the Maximum Packet Length. (default).<br>1 Disable the zero length packet on transfers that are equal in length to a multiple Maximum Packet Length. |
| 28–27 | — | Reserved, should be cleared. These bit reserved for future use and should be cleared. |

**Table 21-74. Endpoint Capabilities/Characteristics (continued)**

| Bits | Name | Description |
|---|---|---|
| 26–16 | Maximum Packet Length | Maximum packet length. This directly corresponds to the maximum packet size of the associated endpoint (wMaxPacketSize). The maximum value this field may contain is 0x400 (1024). |
| 15 | ios | Interrupt on setup (IOS). This bit is used on control type endpoints to indicate if USBINT is set in response to a setup being received. |
| 14–0 | | Reserved, should be cleared. Bits reserved for future use and should be cleared. |

### 21.7.1.2 Transfer Overlay

The seven DWords in the overlay area represent a transaction working space for the device controller. The general operational model is that the device controller can detect whether the overlay area contains a description of an active transfer. If it does not contain an active transfer, then it will not read the associated endpoint.

After an endpoint is readied, the dTD will be copied into this queue head overlay area by the device controller. Until a transfer is expired, software must not write the queue head overlay area or the associated transfer descriptor. When the transfer is complete, the device controller will write the results back to the original transfer descriptor and advance the queue.

See dTD for a description of the overlay fields.

### 21.7.1.3 Current dTD Pointer

The current dTD pointer is used by the device controller to locate the transfer in progress. This word is for the USB controller (hardware) use only and should not be modified by DCD software.

**Table 21-75. Current dTD Pointer**

| Bits | Description |
|---|---|
| 31–5 | Current dtd. This field is a pointer to the dTD that is represented in the transfer overlay area. This field will be modified by the Device Controller to next dTD pointer during endpoint priming or queue advance. |
| 4–0 | Reserved, should be cleared. Bit reserved for future use and should be cleared. |

### 21.7.1.4 Set-up Buffer

The set-up buffer is dedicated storage for the 8-byte data that follows a set-up PID.

**NOTE**

Each endpoint has a TX and an RX dQH associated with it, and only the RX queue head is used for receiving setup data packets.

**Table 21-76. Multiple Mode Control**

| DWord | Bits | Description |
|---|---|---|
| 1 | 31–0 | Setup Buffer 0. This buffer contains bytes 3 to 0 of an incoming setup buffer packet and is written by the device controller to be read by software. |
| 2 | 31–0 | Setup Buffer 1. This buffer contains bytes 7 to 4 of an incoming setup buffer packet and is written by the device controller to be read by software. |

## 21.7.2 Endpoint Transfer Descriptor (dTD)

The dTD describes to the device controller the location and quantity of data to be sent/received for given transfer. The DCD should not attempt to modify any field in an active dTD except the Next Link Pointer, which should only be modified as described in section Managing Transfers with Transfer Descriptors.

<table>
<tr><th>31</th><th>30</th><th>29</th><th>28</th><th>27</th><th>26</th><th>25</th><th>24</th><th>23</th><th>22</th><th>21</th><th>20</th><th>19</th><th>18</th><th>17</th><th>16</th><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th><th>offset</th></tr>
<tr><td colspan="27">Next Link Pointer</td><td colspan="4">0000</td><td>T</td><td>0x00</td></tr>
<tr><td>0</td><td colspan="15">Total Bytes[1]</td><td>ioc</td><td colspan="3">000</td><td colspan="2">MultO</td><td colspan="2">00</td><td colspan="8">Status[1]</td><td>0x04</td></tr>
<tr><td colspan="20">Buffer Pointer (Page 0)</td><td colspan="12">Current Offset[1]</td><td>0x08</td></tr>
<tr><td colspan="20">Buffer Pointer (Page 1)</td><td>0</td><td colspan="11">Frame Number[1]</td><td>0x0C</td></tr>
<tr><td colspan="20">Buffer Pointer (Page 2)</td><td colspan="12">0000_0000_0000</td><td>0x10</td></tr>
<tr><td colspan="20">Buffer Pointer (Page 3)</td><td colspan="12">0000_0000_0000</td><td>0x14</td></tr>
<tr><td colspan="20">Buffer Pointer (Page 4)</td><td colspan="12">0000_0000_0000</td><td>0x18</td></tr>
</table>

**Figure 21-61. Endpoint Transfer Descriptor (dTD)**

[1] Device controller read/write; all others read-only.

**Table 21-77. Next dTD Pointer**

| Bits | Description |
|---|---|
| 31–5 | Next transfer element pointer. This field contains the physical memory address of the next dTD to be processed. The field corresponds to memory address signals [31:5], respectively. |
| 4–1 | Reserved, should be cleared. Bits reserved for future use and should be cleared. |
| 0 | Terminate (T). 1=pointer is invalid. 0=Pointer is valid (points to a valid Transfer Element Descriptor). This bit indicates to the Device Controller that there are no more valid entries in the queue. |

Table 21-78. dTD Token

| Bits | Description |
|---|---|
| 31 | Reserved, should be cleared. Bit reserved for future use and should be cleared. |
| 30–16 | Total Bytes. This field specifies the total number of bytes to be moved with this transfer descriptor. This field is decremented by the number of bytes actually moved during the transaction and only on the successful completion of the transaction.<br>The maximum value software may store in the field is 5*4K(5000H). This is the maximum number of bytes 5 page pointers can access. Although it is possible to create a transfer up to 20K this assumes the 1st offset into the first page is 0. When the offset cannot be predetermined, crossing past the 5th page can be guaranteed by limiting the total bytes to 16K**. Therefore, the maximum recommended transfer is 16K(4000H).<br>If the value of the field is zero when the host controller fetches this transfer descriptor (and the active bit is set), the device controller executes a zero-length transaction and retires the transfer descriptor.<br>It is not a requirement for IN transfers that Total Bytes To Transfer be an even multiple of Maximum Packet Length. If software builds such a transfer descriptor for an IN transfer, the last transaction will always be less that Maximum Packet Length. |
| 15 | Interrupt On Complete (IOC). This bit is used to indicate if USBINT is to be set in response to device controller being finished with this dTD. |
| 14–12 | Reserved, should be cleared. Bits reserved for future use and should be cleared. |
| 11–10 | Multiplier Override (MultO). This field can be used for transmit ISO's (that is, ISO-IN) to override the multiplier in the QH. This field must be zero for all packet types that are not transmit-ISO.<br>Example:<br>if QH.multiplier = 3; Maximum packet size = 8; Total bytes = 15; MultiO = 0 [default]<br>Three packets are sent: {Data2(8); Data1(7); Data0(0)}<br>if QH.multiplier = 3; Maximum packet size = 8; Total bytes = 15; MultiO = 2<br>Two packets are sent: {Data1(8); Data0(7)}<br>For maximal efficiency, software should compute MultO = greatest integer of (Total Bytes/Max. Packet Size) except for the case when Total bytes = 0; then MultO should be 1.<br>**Note:** Non-ISO and non-TX endpoints must set MultO = 00. |
| 9–8 | Reserved, should be cleared. Bits reserved for future use and should be cleared. |
| 7–0 | Status. This field is used by the Device Controller to communicate individual command execution states back to the Device Controller software. This field contains the status of the last transaction performed on this qTD. The bit encodings are:<br>**Bit** **Status Field Description**<br>7 Active<br>6 Halted<br>5 Data Buffer Error<br>3 Transaction Error<br>4,2,0 Reserved, should be cleared |

Table 21-79. Buffer Pointer Page 0

| Bits | Description |
|---|---|
| 31–12 | Buffer Pointer. Selects the page offset in memory for the packet buffer. Non virtual memory systems will typically set the buffer pointers to a series of incrementing integers. |
| 11–0 | Current Offset. Offset into the 4kb buffer where the packet is to begin. |

Table 21-80. Buffer Pointer Page 1

| Bits | Description |
|---|---|
| 31–12 | Buffer Pointer. Selects the page offset in memory for the packet buffer. Non virtual memory systems will typically set the buffer pointers to a series of incrementing integers. |
| 11 | Reserved |
| 10–0 | Frame Number. Written by the device controller to indicate the frame number in which a packet finishes. This is typically be used to correlate relative completion times of packets on an ISO endpoint. |

Table 21-81. Buffer Pointer Pages 2–4

| Bits | Description |
|---|---|
| 31–12 | Buffer Pointer. Selects the page offset in memory for the packet buffer. Non virtual memory systems will typically set the buffer pointers to a series of incrementing integers. |
| 11–0 | Reserved |

## 21.8 Device Operational Model

The function of the device operation is to transfer a request in the memory image to and from the Universal Serial Bus. Using a set of linked list transfer descriptors, pointed to by a queue head, the device controller will perform the data transfers. The following sections explain the use of the device controller from the device controller driver (DCD) point-of-view and further describe how specific USB bus events relate to status changes in the device controller programmer's interface.

### 21.8.1 Device Controller Initialization

After hardware reset, the USB DR module is disabled until the run/stop bit (USBCMD[RS]) is set to a '1'. In the disabled state, the pull-up on the USB D+ is not active which prevents an attach event from occurring. At a minimum, it is necessary to have the queue heads setup for endpoint zero before the device attach occurs. Shortly after the device is enabled, a USB reset will occur followed by setup packet arriving at endpoint 0. A queue head must be prepared so that the device controller can store the incoming setup packet.

In order to initialize a device, the software should perform the following steps:

1. Set the controller mode to device mode. Optionally set USBMODE[SDIS] (streaming disable).

   **NOTE**

   Transitioning from host mode to device mode requires a device controller reset before modifying USBMODE.

2. Optionally modify the BURSTSIZE register.
3. Program PORTSC[PTS] if using a non-ULPI PHY.
4. Set CONTROL[USB_EN]
5. Allocate and initialize device queue heads in system memory Minimum: Initialize device queue heads 0 Tx and 0 Rx.

**NOTE**

> All device queue heads must be initialized for control endpoints before the endpoint is enabled. Device queue heads for non-control endpoints must be initialized before the endpoint can be used.

For information on device queue heads, refer toSection 21.7, “Device Data Structures.”

6. Configure the ENDPOINTLISTADDR pointer.

   For additional information on ENDPOINTLISTADDR, refer to the register table.

7. Enable the microprocessor interrupt associated with the USB module and optionally change setting of USBCMD[ITC].

   Recommended: enable all device interrupts including: USBINT, USBERRINT, Port Change Detect, USB Reset Received, DCSuspend.

   For a list of available interrupts refer to the USBINTR and the USBSTS register tables.

8. Set USBCMD[RS] to run mode.

   After the run bit is set, a device reset will occur. The DCD must monitor the reset event and set the DEVICEADDR register, set the ENDPTCTRLx registers, and adjust the software state as described in the Bus Reset section of the following Port State and Control section below.

**NOTE**

> Endpoint 0 is designed as a control endpoint only and does not need to be configured using ENDPTCTRL0 register.

It is also not necessary to initially prime Endpoint 0 because the first packet received will always be a setup packet. The contents of the first setup packet will require a response in accordance with USB device framework command set.

### 21.8.2 Port State and Control

From a chip or system reset, the USB controller enters the powered state. A transition from the powered state to the attach state occurs when the run/stop bit (USBCMD[RS]) is set to a '1'. After receiving a reset on the bus, the port will enter the defaultFS or defaultHS state in accordance with the protocol reset described in Appendix C.2 of the USB Specification Rev. 2.0. The following state diagram depicts the state of a USB 2.0 device.

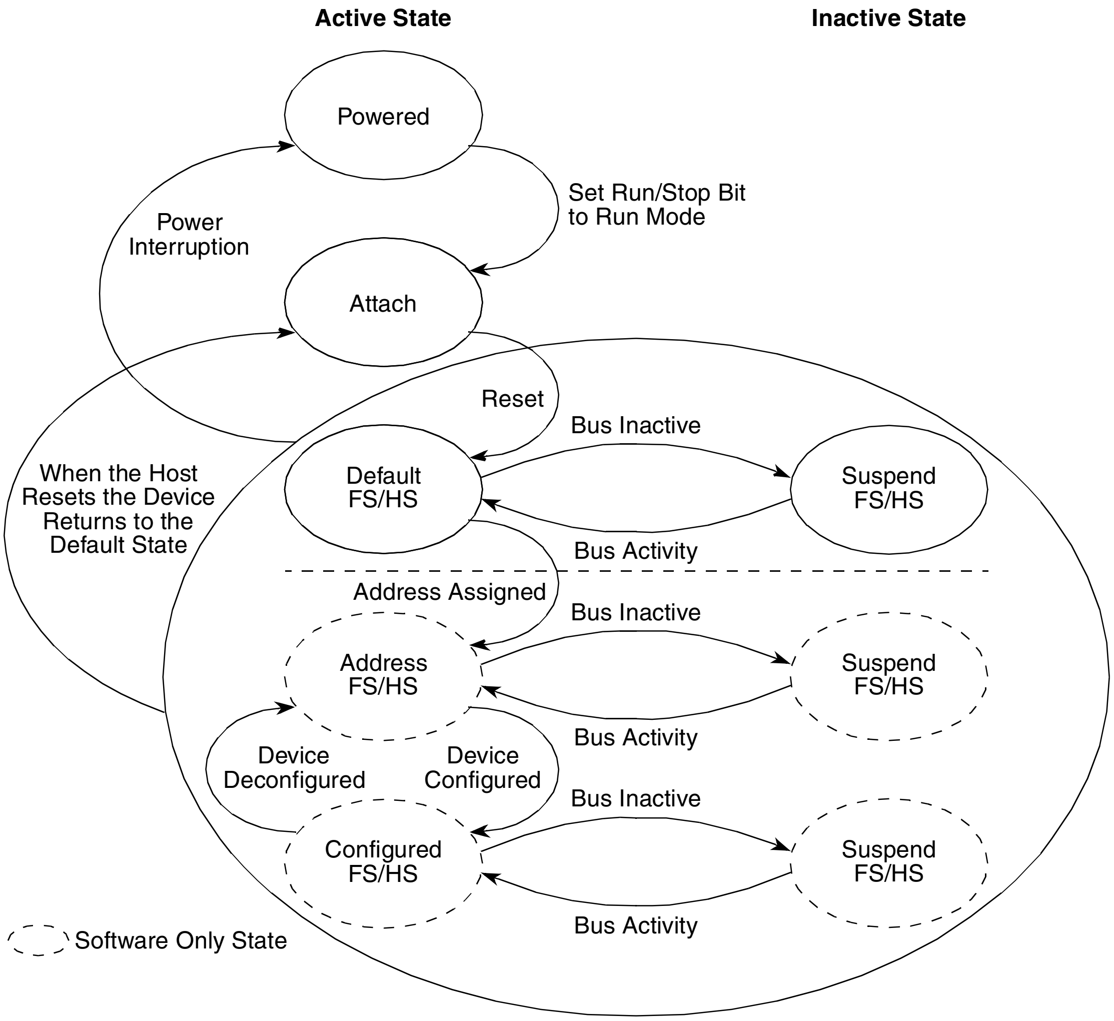


**Figure 21-62. USB 2.0 Device States**

States powered, attach, defaultFS/HS, suspendFS/HS are implemented in the USB controller and are communicated to the DCD using the following status bits:

**Table 21-82. Device Controller State Information Bits**

| Bits | Register |
|---|---|
| DCSuspend (SLI) | USBSTS |
| USB Reset Received (URI) | USBSTS |
| Port Change Detect (PCI) | USBSTS |
| High-Speed Port | PORTSC |

It is the responsibility of the DCD to maintain a state variable to differentiate between the DefaultFS/HS state and the Address/Configured states. Change of state from Default to Address and the Configured states is part of the enumeration process described in the device framework section of the USB 2.0 Specification.

As a result of entering the Address state, the device address register (DEVICEADDR) must be programmed by the DCD.

Entry into the Configured indicates that all endpoints to be used in the operation of the device have been properly initialized by programming the ENDPTCTRL*n* registers and initializing the associated queue heads.

### 21.8.2.1 Bus Reset

A bus reset is used by the host to initialize downstream devices. When a bus reset is detected, the USB controller will renegotiate its attachment speed, reset the device address to 0, and notify the DCD by interrupt (assuming the USB reset interrupt enable bit, USBINTR[URE], is set). After a reset is received, all endpoints (except endpoint 0) are disabled and any primed transactions will be cancelled by the device controller. The concept of priming will be clarified below, but the DCD must perform the following tasks when a reset is received:

Clear all setup token semaphores by reading the ENDPTSETUPSTAT register and writing the same value back to the ENDPTSETUPSTAT register.

Clear all the endpoint complete status bits by reading the ENDPTCOMPLETE register and writing the same value back to the ENDPTCOMPLETE register.

Cancel all primed status by waiting until all bits in the ENDPTPRIME are 0 and then writing 0xFFFF_FFFF to ENDPTFLUSH.

Read the reset bit in the PORTSC register (PORTSC[PR]) and make sure that it is still active. A USB reset will occur for a minimum of 3 ms and the DCD must reach this point in the reset cleanup before end of the reset occurs, otherwise a hardware reset of the device controller is recommended (rare.)

- A hardware reset can be performed by writing a one to the USB reset bit in (USBCMD[RST]). Note: a hardware reset will cause the device to detach from the bus by clearing USBCMD[RS] bit. Thus, the DCD must completely re-initialize the USB after a hardware reset.

Free all allocated dTDs because they will no longer be executed by the device controller. If this is the first time the DCD is processing a USB reset event, then it is likely that no dTDs have been allocated.

At this time, the DCD may release control back to the OS because no further changes to the device controller are permitted until a Port Change Detect is indicated.

After a Port Change Detect, the device has reached the default state and the DCD can read the PORTSC to determine if the device is operating in FS or HS mode. At this time, the device controller has reached normal operating mode and DCD can begin enumeration according to the USB Chapter 9 - Device Framework.

**NOTE**

> The device DCD may use the FS/HS mode information to determine the bandwidth mode of the device.

In some applications, it may not be possible to enable one or more pipes while in FS mode. Beyond the data rate issue, there is no difference in DCD operation between FS and HS modes.

### 21.8.2.2 Suspend/Resume

#### 21.8.2.2.1 Suspend Description

In order to conserve power, USB controller automatically enters the suspended state when no bus traffic has been observed for a specified period. When suspended, the USB controller maintains any internal status, including its address and configuration. Attached devices must be prepared to suspend at any time they are powered, regardless of if they have been assigned a non-default address, are configured, or neither. Bus activity may cease due to the host entering a suspend mode of its own. In addition, a USB device shall also enter the suspended state when the hub port it is attached to is disabled.

The USB controller exits suspend mode when there is bus activity. It may also request the host to exit suspend mode or selective suspend by using electrical signaling to indicate remote wake-up. The ability of a device to signal remote wake-up is optional. The USB controller is capable of remote wake-up signaling. When the USB controller is reset, remote wake-up signaling must be disabled.

#### 21.8.2.2.2 Suspend Operational Model

The USB controller moves into the suspend state when suspend signaling is detected or activity is missing on the upstream port for more than a specific period. After the device controller enters the suspend state, the DCD is notified by an interrupt (assuming DC Suspend Interrupt is enabled). When the USBSTS[SLI] (device controller suspend) is set, the device controller is suspended.

DCD response when the device controller is suspended is application specific and may involve switching to low power operation.

Information on the bus power limits in suspend state can be found in USB 2.0 specification.

#### 21.8.2.2.3 Resume

If the USB controller is suspended, its operation is resumed when any non-idle signaling is received on its upstream facing port. In addition, the USB controller can signal the system to resume operation by forcing resume signaling to the upstream port. Resume signaling is sent upstream by writing a '1' to the PORTSC[FPR] (resume bit) while the device is in suspend state. Sending resume signal to an upstream port should cause the host to issue resume signaling and bring the suspended bus segment (one more devices) back to the active condition.

**NOTE**

> Before resume signaling can be used, the host must enable it by using the Set Feature command defined in device framework (Chapter 9) of the USB 2.0 Specification.

### 21.8.3 Managing Endpoints

The USB 2.0 specification defines an endpoint, also called a device endpoint or an address endpoint as a uniquely addressable portion of a USB device that can source or sink data in a communications channel between the host and the device. The endpoint address is specified by the combination of the endpoint number and the endpoint direction.

The channel between the host and an endpoint at a specific device represents a data pipe. Endpoint 0 for a device is always a control type data channel used for device discovery and enumeration. Other types of endpoints support by USB include bulk, interrupt, and isochronous. Each endpoint type has specific behavior related to packet response and error handling. More detail on endpoint operation can be found in the USB 2.0 specification.

The USB controller supports up to six endpoint specified numbers. The DCD can enable, disable and configure each endpoint.

Each endpoint direction is essentially independent and can be configured with differing behavior in each direction. For example, the DCD can configure endpoint 1-IN to be a bulk endpoint and endpoint 1-OUT to be an isochronous endpoint. This helps to conserve the total number of endpoints required for device operation. The only exception is that control endpoints must use both directions on a single endpoint number to function as a control endpoint. Endpoint 0 is, for example, is always a control endpoint and uses the pair of directions.

Each endpoint direction requires a queue head allocated in memory. If the maximum of 6 endpoint numbers, one for each endpoint direction are being used by the device controller, then 12 queue heads are required. The operation of an endpoint and use of queue heads are described later in this document.

### 21.8.3.1 Endpoint Initialization

After hardware reset, all endpoints except endpoint zero are uninitialized and disabled. The DCD must configure and enable each endpoint by writing to configuration bit in the ENDPTCTRL*n* register. Each 32-bit ENDPTCTRL*n* is split into an upper and lower half. The lower half of ENDPTCTRL*n* is used to configure the receive or OUT endpoint and the upper half is likewise used to configure the corresponding transmit or IN endpoint. Control endpoints must be configured the same in both the upper and lower half of the ENDPTCTRL*n* register otherwise the behavior is undefined. The following table shows how to construct a configuration word for endpoint initialization.

**Table 21-83. Device Controller Endpoint Initialization**

| Field | Value |
|---|---|
| Data Toggle Reset | 1 |
| Data Toggle Inhibit | 0 |
| Endpoint Type | 00 Control<br>01 Isochronous<br>10 Bulk<br>11 Interrupt |
| Endpoint Stall | 0 |

#### 21.8.3.1.1 Stalling

There are two occasions where the USB controller may need to return to the host a STALL.

The first occasion is the functional stall, which is a condition set by the DCD as described in the USB 2.0 device framework (chapter 9). A functional stall is only used on non-control endpoints and can be enabled in the device controller by setting the endpoint stall bit in the ENDPTCTRL*n* register associated with the given endpoint and the given direction. In a functional stall condition, the device controller will continue to return STALL responses to all transactions occurring on the respective endpoint and direction until the endpoint stall bit is cleared by the DCD.

A protocol stall, unlike a function stall, is used on control endpoints is automatically cleared by the device controller at the start of a new control transaction (setup phase). When enabling a protocol stall, the DCD should enable the stall bits (both directions) as a pair. A single write to the ENDPTCTRL*n* register can ensure that both stall bits are set at the same instant.

**NOTE**

> Any write to the ENDPTCTRL*n* register during operational mode must preserve the endpoint type field (that is, perform a read-modify-write).

**Table 21-84. Device Controller Stall Response Matrix**

| USB Packet | Endpoint Stall Bit. | Effect on STALL Bit. | USB Response |
|---|---|---|---|
| SETUP packet received by a non-control endpoint | N/A | None | STALL |
| IN/OUT/PING packet received by a non-control endpoint | '1 | None | STALL |
| IN/OUT/PING packet received by a non-control endpoint | '0 | None | ACK/NAK/NYET |
| SETUP packet received by a control endpoint | N/A | Cleared | ACK |
| IN/OUT/PING packet received by a control endpoint | '1 | None | STALL |
| IN/OUT/PING packet received by a control endpoint | '0 | None | ACK/NAK/NYET |

### 21.8.3.2 Data Toggle

Data toggle is a mechanism to maintain data coherency between host and device for any given data pipe. For more information on data toggle, refer to the *Universal Serial Bus Revision 2.0 Specification*.

#### 21.8.3.2.1 Data Toggle Reset

The DCD may reset the data toggle state bit and cause the data toggle sequence to reset in the device controller by writing a '1' to the data toggle reset bit in the ENDPTCTRL*n* register. This should only be necessary when configuring/initializing an endpoint or returning from a STALL condition.

#### 21.8.3.2.2 Data Toggle Inhibit

This feature is for test purposes only and should never be used during normal device controller operation.

Setting the data toggle Inhibit bit active ('1') causes the USB controller to ignore the data toggle pattern that is normally sent and accept all incoming data packets regardless of the data toggle state.

In normal operation, the USB controller checks the DATA0/DATA1 bit against the data toggle to determine if the packet is valid. If Data PID does not match the data toggle state bit maintained by the device controller for that endpoint, the Data toggle is considered not valid. If the data toggle is not valid, the device controller assumes the packet was already received and discards the packet (not reporting it to the

DCD). To prevent the USB controller from re-sending the same packet, the device controller will respond to the error packet by acknowledging it with either an ACK or NYET response.

### 21.8.3.3 Device Operational Model For Packet Transfers

All transactions on the USB bus are initiated by the host and in turn, the device must respond to any request from the host within the turnaround time stated in the *Universal Serial Bus Revision 2.0 Specification*.

A USB host will send requests to the USB controller in an order that can not be precisely predicted as a single pipeline, so it is not possible to prepare a single packet for the device controller to execute. However, the order of packet requests is predictable when the endpoint number and direction is considered. For example, if endpoint 2 (transmit direction) is configured as a bulk pipe, then we can expect the host will send IN requests to that endpoint. This USB controller prepares packets for each endpoint/direction in anticipation of the host request. The process of preparing the device controller to send or receive data in response to host initiated transaction on the bus is referred to as ‘priming’ the endpoint. This term will be used throughout the following documentation to describe the USB controller operation so the DCD can be architected properly use priming. Further, note that the term ‘flushing’ is used to describe the action of clearing a packet that was queued for execution.

#### 21.8.3.3.1 Priming Transmit Endpoints

Priming a transmit endpoint will cause the device controller to fetch the device transfer descriptor (dTD) for the transaction pointed to by the device queue head (dQH). After the dTD is fetched, it will be stored in the dQH until the device controller completes the transfer described by the dTD. Storing the dTD in the dQH allows the device controller to fetch the operating context needed to handle a request from the host without the need to follow the linked list, starting at the dQH when the host request is received.

After the device has loaded the dTD, the leading data in the packet is stored in a FIFO in the device controller. This FIFO is split into virtual channels so that the leading data can be stored for any endpoint up to the maximum number of endpoints configured at device synthesis time.

After a priming request is complete, an endpoint state of primed is indicated in the ENDPTSTATUS register. For a primed transmit endpoint, the device controller can respond to an IN request from the host and meet the stringent bus turnaround time of High Speed USB.

Since only the leading data is stored in the device controller FIFO, it is necessary for the device controller to begin filling in behind leading data after the transaction starts. The FIFO must be sized to account for the maximum latency that can be incurred by the system memory bus.

#### 21.8.3.3.2 Priming Receive Endpoints

Priming receive endpoints is identical to priming of transmit endpoints from the point of view of the DCD. At the device controller the major difference in the operational model is that there is no data movement of the leading packet data simply because the data is to be received from the host.

Note as part of the architecture, the FIFO for the receive endpoints is not partitioned into multiple channels like the transmit FIFO. Thus, the size of the RX FIFO does not scale with the number of endpoints.

#### 21.8.3.4 Interrupt/Bulk Endpoint Operational Model

The behaviors of the device controller for interrupt and bulk endpoints are identical. All valid IN and OUT transactions to bulk pipes will handshake with a NAK unless the endpoint had been primed. Once the endpoint has been primed, data delivery will commence.

A dTD will be retired by the device controller when the packets described in the transfer descriptor have been completed. Each dTD describes N packets to be transferred according to the USB Variable Length transfer protocol. The formula and table on the following page describe how the device controller computes the number and length of the packets to be sent/received by the USB vary according to the total number of bytes and maximum packet length.

With Zero Length Termination (ZLT) = 0

N = INT(number of bytes/max. packet length) + 1

With Zero Length Termination (ZLT) = 1

N = MAXINT(number of bytes/max. packet length)

**Table 21-85. Variable Length Transfer Protocol Example (ZLT = 0)**

| Bytes (dTD) | Max. Packet Length (dQH) | N | P1 | P2 | P3 |
|---|---|---|---|---|---|
| 511 | 256 | 2 | 256 | 255 | |
| 512 | 256 | 3 | 256 | 256 | 0 |
| 512 | 512 | 2 | 512 | 0 | |

**Table 21-86. Variable Length Transfer Protocol Example (ZLT = 1)**

| Bytes (dTD) | Max. Packet Length (dQH) | N | P1 | P2 | P3 |
|---|---|---|---|---|---|
| 511 | 256 | 2 | 256 | 255 | |
| 512 | 256 | 2 | 256 | 256 | |
| 512 | 512 | 1 | 512 | | |

**NOTE**

The MULT field in the dQH must be set to ‘00’ for bulk, interrupt, and control endpoints.

TX-dTD is complete when:

- All packets described dTD were successfully transmitted. *** Total bytes in dTD will equal zero when this occurs.

RX-dTD is complete when:

- All packets described in dTD were successfully received. *** Total bytes in dTD will equal zero when this occurs.

- A short packet (number of bytes < maximum packet length) was received. *** This is a successful transfer completion; DCD must check Total Bytes in dTD to determine the number of bytes that are remaining. From the total bytes remaining in the dTD, the DCD can compute the actual bytes received.
- A long packet was received (number of bytes > maximum packet size) OR (total bytes received > total bytes specified). *** This is an error condition. The device controller will discard the remaining packet, and set the Buffer Error bit in the dTD. In addition, the endpoint will be flushed and the USBERR interrupt will become active.

On the successful completion of the packet(s) described by the dTD, the active bit in the dTD will be cleared and the next pointer will be followed when the Terminate bit is clear. When the Terminate bit is set, the USB controller will flush the endpoint/direction and cease operations for that endpoint/direction.

On the unsuccessful completion of a packet (see long packet above), the dQH will be left pointing to the dTD that was in error. In order to recover from this error condition, the DCD must properly re-initialize the dQH by clearing the active bit and update the nextTD pointer before attempting to re-prime the endpoint.

**NOTE**

All packet level errors such as a missing handshake or CRC error will be retried automatically by the device controller.

There is no required interaction with the DCD for handling such errors.

#### 21.8.3.4.1 Interrupt/Bulk Endpoint Bus Response Matrix

**Table 21-87. Interrupt/Bulk Endpoint Bus Response Matrix**

| | Stall | Not Primed | Primed | Underflow | Overflow |
|---|---|---|---|---|---|
| **Setup** | Ignore | Ignore | Ignore | N/A | N/A |
| **In** | STALL | NAK | Transmit | BS Error[1] | N/A |
| **Out** | STALL | NAK | Receive + NYET/ACK[2] | N/A | NAK |
| **Ping** | STALL | NAK | ACK | N/A | N/A |
| **Invalid** | Ignore | Ignore | Ignore | Ignore | Ignore |

[1] Force Bit Stuff Error.

[2] NYET/ACK—NYET unless the Transfer Descriptor has packets remaining according to the USB variable length protocol then ACK.
SYSERR—System error should never occur when the latency FIFOs are correctly sized and the DCD is responsive.

### 21.8.3.5 Control Endpoint Operation Model

#### 21.8.3.5.1 Setup Phase

All requests to a control endpoint begin with a setup phase followed by an optional data phase and a required status phase. The USB controller will always accept the setup phase unless the setup lockout is engaged.

The setup lockout will engage so that future setup packets are ignored. Lockout of setup packets ensures that while software is reading the setup packet stored in the queue head, that data is not written as it is being read potentially causing an invalid setup packet.

The setup lockout mechanism can be disabled and a tripwire type semaphore will ensure that the setup packet payload is extracted from the queue head without being corrupted be an incoming setup packet. This is the preferred behavior because ignoring repeated setup packets due to long software interrupt latency would be a compliance issue.

Setup Packet Handling

- Disable Setup Lockout by writing '1' to Setup Lockout Mode (SLOM) in USBMODE. (once at initialization). Setup lockout is not necessary when using the tripwire as described below.

**NOTE**

> Leaving the Setup Lockout Mode As '0' will result in a potential compliance issue.

- After receiving an interrupt and inspecting ENDPTSETUPSTAT to determine that a setup packet was received on a particular pipe:
  - — Write ‘1’ to clear corresponding bit ENDPTSETUPSTAT.
  - — Write ‘1’ to Setup Tripwire (SUTW) in USBCMD register.
  - — Duplicate contents of dQH.SetupBuffer into local software byte array.
  - — Read Setup TripWire (SUTW) in USBCMD register. (if set – continue; if cleared – goto 2)
  - — Write ‘0’ to clear Setup Tripwire (SUTW) in USBCMD register.
  - — Process setup packet using local software byte array copy and execute status/handshake phases.

  Note: After receiving a new setup packet the status and/or handshake phases may still be pending from a previous control sequence. These should be flushed and de-allocated before linking a new status and/or handshake dTD for the most recent setup packet.

#### 21.8.3.5.2 Data Phase

Following the setup phase, the DCD must create a device transfer descriptor for the data phase and prime the transfer.

After priming the packet, the DCD must verify a new setup packet has not been received by reading the ENDPTSETUPSTAT register immediately verifying that the prime had completed. A prime will complete when the associated bit in the ENDPTPRIME register is zero and the associated bit in the ENDPTSTATUS register is a one. If a prime fails, that is, The ENDPTPRIME bit goes to zero and the ENDPTSTATUS bit is not set, then the prime has failed. This can only be due to improper setup of the dQH, dTD or a setup arriving during the prime operation. If a new setup packet is indicated after the ENDPTPRIME bit is cleared, then the transfer descriptor can be freed and the DCD must reinterpret the setup packet.

Should a setup arrive after the data stage is primed, the device controller will automatically clear the prime status (ENDPTSTATUS) to enforce data coherency with the setup packet.

**NOTE**

The MULT field in the dQH must be set to ‘00’ for bulk, interrupt, and control endpoints.

**NOTE**

Error handling of data phase packets is the same as bulk packets described previously.

#### 21.8.3.5.3 Status Phase

Similar to the data phase, the DCD must create a transfer descriptor (with byte length equal zero) and prime the endpoint for the status phase. The DCD must also perform the same checks of the ENDPTSETUPSTAT as described above in the data phase.

**NOTE**

The MULT field in the dQH must be set to ‘00’ for bulk, interrupt, and control endpoints.

**NOTE**

Error handling of data phase packets is the same as bulk packets described previously.

#### 21.8.3.5.4 Control Endpoint Bus Response Matrix

Shown in the following table is the device controller response to packets on a control endpoint according to the device controller state.

**Table 21-88. Control Endpoint Bus Response Matrix**

| Token Type | Endpoint State | | | | | Setup Lockout |
|---|---|---|---|---|---|---|
| | Stall | Not Primed | Primed | Underflow | Overflow | |
| Setup | ACK | ACK | ACK | N/A | SYSERR[1] | |
| In | STALL | NAK | Transmit | BS Error[2] | N/A | N/A |
| Out | STALL | NAK | Receive + NYET/ACK[3] | N/A | NAK | N/A |

Table 21-88. Control Endpoint Bus Response Matrix (continued)

| Token Type | Endpoint State | | | | | Setup Lockout |
|---|---|---|---|---|---|---|
| | Stall | Not Primed | Primed | Underflow | Overflow | |
| **Ping** | STALL | NAK | ACK | N/A | N/A | N/A |
| **Invalid** | Ignore | Ignore | Ignore | Ignore | Ignore | Ignore |

[1] SYSERR—System error should never occur when the latency FIFOs are correctly sized and the DCD is responsive.
[2] Force Bit Stuff Error.
[3] NYET/ACK—NYET unless the Transfer Descriptor has packets remaining according to the USB variable length protocol then ACK.

### 21.8.3.6 Isochronous Endpoint Operational Model

Isochronous endpoints are used for real-time scheduled delivery of data and their operational model is significantly different than the host throttled Bulk, Interrupt, and Control data pipes. Real time delivery by the USB controller will is accomplished by the following:

- Exactly MULT Packets per (micro)Frame are transmitted/received. Note: MULT is a two-bit field in the device Queue Head. The variable length packet protocol is not used on isochronous endpoints.
- NAK responses are not used. Instead, zero length packets and sent in response to an IN request to an unprimed endpoints. For unprimed RX endpoints, the response to an OUT transaction is to ignore the packet within the device controller.
- Prime requests always schedule the transfer described in the dTD for the next (micro)frame. If the ISO-dTD is still active after that frame, then the ISO-dTD will be held ready until executed or canceled by the DCD.

The USB controller in host mode uses the periodic frame list to schedule data exchanges to Isochronous endpoints. The operational model for device mode does not use such a data structure. Instead, the same dTD used for Control/Bulk/Interrupt endpoints is also used for isochronous endpoints. The difference is in the handling of the dTD.

The first difference between bulk and ISO-endpoints is that priming an ISO-endpoint is a delayed operation such that an endpoint will become primed only after a SOF is received. After the DCD writes the prime bit, the prime bit will be cleared as usual to indicate to software that the device controller completed a priming the dTD for transfer. Internal to the design, the device controller hardware masks that prime start until the next frame boundary. This behavior is hidden from the DCD but occurs so that the device controller can match the dTD to a specific (micro) frame.

Another difference with isochronous endpoints is that the transaction must wholly complete in a (micro)frame. Once an ISO transaction is started in a (micro)frame it will retire the corresponding dTD when MULT transactions occur or the device controller finds a fulfillment condition.

The transaction error bit set in the status field indicates a fulfillment error condition. When a fulfillment error occurs, the frame after the transfer failed to complete wholly, the device controller will force retire the ISO-dTD and move to the next ISO-dTD.

It is important to note that fulfillment errors are only caused due to partially completed packets. If no activity occurs to a primed ISO-dTD, the transaction will stay primed indefinitely. This means it is up to software discard transmit ISO-dTDs that pile up from a failure of the host to move the data.

Finally, the last difference with ISO packets is in the data level error handling. When a CRC error occurs on a received packet, the packet is not retried similar to bulk and control endpoints. Instead, the CRC is noted by setting the Transaction Error bit and the data is stored as usual for the application software to sort out.

- TX Packet Retired
  - MULT counter reaches zero.
  - Fulfillment Error [Transaction Error bit is set]
  - #Packets Occurred > 0 AND # Packets Occurred < MULT

**NOTE**

For TX-ISO, MULT Counter can be loaded with a lesser value in the dTD Multiplier Override field. If the Multiplier Override is zero, the MULT Counter is initialized to the Multiplier in the QH.

- RX Packet Retired:
  - MULT counter reaches zero.
  - Non-MDATA Data PID is received
  - Overflow Error:
  - Packet received is > maximum packet length. [Buffer Error bit is set]
  - Packet received exceeds total bytes allocated in dTD. [Buffer Error bit is set]
  - Fulfillment Error [Transaction Error bit is set]
  - # Packets Occurred > 0 AND # Packets Occurred < MULT
  - CRC Error [Transaction Error bit is set]

**NOTE**

For ISO, when a dTD is retired, the next dTD is primed for the next frame. For continuous (micro)frame to (micro)frame operation the DCD should ensure that the dTD linked-list is out ahead of the device controller by at least two (micro)frames.

#### 21.8.3.6.1 Isochronous Pipe Synchronization

When it is necessary to synchronize an isochronous data pipe to the host, the (micro)frame number (FRINDEX register) can be used as a marker. To cause a packet transfer to occur at a specific (micro)frame number [N], the DCD should interrupt on SOF during frame N-1. When the FRINDEX=N-1, the DCD must write the prime bit. The USB controller will prime the isochronous endpoint in (micro)frame N-1 so that the device controller will execute delivery during (micro)frame N.

**CAUTION**

Priming an endpoint towards the end of (micro)frame N-1 will not guarantee delivery in (micro)frame N. The delivery may actually occur in (micro)frame N+1 if device controller does not have enough time to complete the prime before the SOF for packet N is received.

### 21.8.3.6.2 Isochronous Endpoint Bus Response Matrix

**Table 21-89. Isochronous Endpoint Bus Response Matrix**

| | Stall | Not Primed | Primed | Underflow | Overflow |
|---|---|---|---|---|---|
| **Setup** | STALL | STALL | STALL | N/A | N/A |
| **In** | NULL[1] Packet | NULL Packet | Transmit | BS Error[2] | N/A |
| **Out** | Ignore | Ignore | Receive | N/A | Drop Packet |
| **Ping** | Ignore | Ignore | Ignore | Ignore | Ignore |
| **Invalid** | Ignore | Ignore | Ignore | Ignore | Ignore |

[1] Zero Length Packet.
[2] Force Bit Stuff Error.

## 21.8.4 Managing Queue Heads

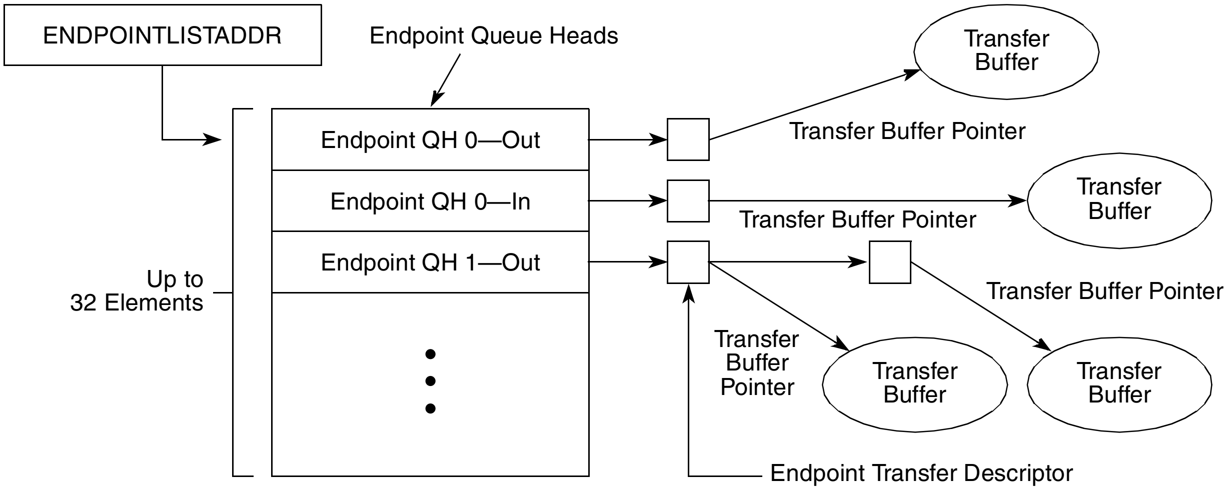


**Figure 21-63. Endpoint Queue Head Diagram**

The device queue head (dQH) points to the linked list of transfer tasks, each depicted by the device Transfer Descriptor (dTD). An area of memory pointed to by ENDPOINTLISTADDR contains a group of all dQH's in a sequential list as shown in Figure 21-63. The even elements in the list of dQH's are used for receive endpoints (OUT/SETUP) and the odd elements are used for transmit endpoints (IN/INTERRUPT). Device transfer descriptors are linked head to tail starting at the queue head and ending at a terminate bit. Once the dTD has been retired, it will no longer be part of the linked list from the queue head. Therefore, software is required to track all transfer descriptors since pointers will no longer exist within the queue head once the dTD is retired (see section Software Link Pointers).

In addition to the current and next pointers and the dTD overlay examined in section Operational Model For Packet Transfers, the dQH also contains the following parameters for the associated endpoint: Multipler, Maximum Packet Length, Interrupt On Setup. The complete initialization of the dQH including these fields is demonstrated in the next section.

#### 21.8.4.1 Queue Head Initialization

One pair of device queue heads must be initialized for each active endpoint. To initialize a device queue head:

- Write the wMaxPacketSize field as required by the USB Chapter 9 or application specific protocol.
- Write the multiplier field to 0 for control, bulk, and interrupt endpoints. For ISO endpoints, set the multiplier to 1,2, or 3 as required bandwidth an in conjunction with the USB Chapter 9 protocol. Note: In FS mode, the multiplier field can only be 1 for ISO endpoints.
- Write the next dTD Terminate bit field to ‘1.’
- Write the Active bit in the status field to ‘0.’
- Write the Halt bit in the status field to ‘0.’

**NOTE**

The DCD must only modify dQH if the associated endpoint is not primed and there are no outstanding dTDs.

#### 21.8.4.2 Operational Model for Setup Transfers

As discussed in Section 21.8.3.5, “Control Endpoint Operation Model,” setup transfer requires special treatment by the DCD. A setup transfer does not use a dTD but instead stores the incoming data from a setup packet in an 8-byte buffer within the dQH.

Upon receiving notification of the setup packet, the DCD should handle the setup transfer as demonstrated here:

1. Copy setup buffer contents from dQH - RX to software buffer.
2. Acknowledge setup backup by writing a ‘1’ to the corresponding bit in ENDPTSETUPSTAT.

**NOTE**

The acknowledge must occur before continuing to process the setup packet.

**NOTE**

After the acknowledge has occurred, the DCD must not attempt to access the setup buffer in the dQH - RX. Only the local software copy should be examined.

3. Check for pending data or status dTD's from previous control transfers and flush if any exist as discussed in section Flushing/De-priming an Endpoint.

**NOTE**

> It is possible for the device controller to receive setup packets before previous control transfers complete. Existing control packets in progress must be flushed and the new control packet completed.

4. Decode setup packet and prepare data phase [optional] and status phase transfer as required by the USB Chapter 9 or application specific protocol.

## 21.8.5 Managing Transfers with Transfer Descriptors

### 21.8.5.1 Software Link Pointers

It is necessary for the DCD software to maintain head and tail pointers to the for the linked list of dTDs for each respective queue head. This is necessary because the dQH only maintains pointers to the current working dTD and the next dTD to be executed. The operations described in next section for managing dTD will assume the DCD can use reference the head and tail of the dTD linked list.

**NOTE**

> To conserve memory, the reserved fields at the end of the dQH can be used to store the Head and Tail pointers but it still remains the responsibility of the DCD to maintain the pointers.

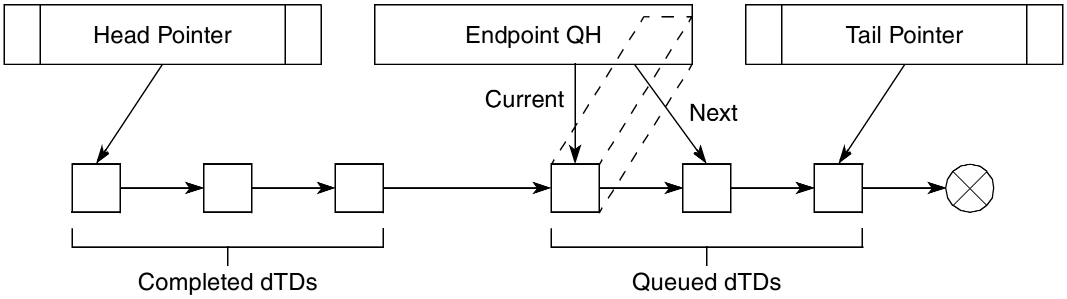


**Figure 21-64. Software Link Pointers**

### 21.8.5.2 Building a Transfer Descriptor

Before a transfer can be executed from the linked list, a dTD must be built to describe the transfer. Use the following procedure for building dTDs.

Allocate 8-DWord dTD block of memory aligned to 8-DWord boundaries. Example: bit address 4–0 would be equal to ‘00000’

Write the following fields:

1. Initialize first 7 DWords to 0.
2. Set the terminate bit to ‘1.’
3. Fill in total bytes with transfer size.
4. Set the interrupt on complete if desired.
5. Initialize the status field with the active bit set to ‘1’ and all remaining status bits set to ‘0.’

6. Fill in buffer pointer page 0 and the current offset to point to the start of the data buffer.
7. Initialize buffer pointer page 1 through page 4 to be one greater than each of the previous buffer pointer.

#### 21.8.5.3 Executing a Transfer Descriptor

To safely add a dTD, the DCD must be follow this procedure which will handle the event where the device controller reaches the end of the dTD list at the same time a new dTD is being added to the end of the list.

Determine whether the link list is empty:

> Check DCD driver to see if pipe is empty (internal representation of linked-list should indicate if any packets are outstanding)

Case 1: Link list is empty

1. Write dQH next pointer AND dQH terminate bit to 0 as a single DWord operation.
2. Clear active and halt bit in dQH (in case set from a previous error).
3. Prime endpoint by writing ‘1’ to correct bit position in ENDPTPRIME.

Case 2: Link list is not empty

1. Add dTD to end of linked list.
2. Read correct prime bit in ENDPTPRIME - if ‘1’ DONE.
3. Set ATDTW bit in USBCMD register to ‘1.’
4. Read correct status bit in ENDPTSTATUS. (store in tmp. variable for later)
5. Read ATDTW bit in USBCMD register.

   If ‘0’ goto 3.

   If '1' continue to 6.
6. Write ATDTW bit in USBCMD register to ‘0.’
7. If status bit read in (3) is ‘1’ DONE.
8. If status bit read in (3) is ‘0’ then Goto Case 1: Step 1.

#### 21.8.5.4 Transfer Completion

After a dTD has been initialized and the associated endpoint primed the device controller will execute the transfer upon the host-initiated request. The DCD will be notified with a USB interrupt if the Interrupt On Complete bit was set or alternately, the DCD can poll the endpoint complete register to find when the dTD had been executed. After a dTD has been executed, DCD can check the status bits to determine success or failure.

**CAUTION**

> Multiple dTD can be completed in a single endpoint complete notification. After clearing the notification, DCD must search the dTD linked list and retire all dTDs that have finished (Active bit cleared).

By reading the status fields of the completed dTDs, the DCD can determine if the transfers completed successfully. Success is determined with the following combination of status bits:

- Active = 0
- Halted = 0
- Transaction Error = 0
- Data Buffer Error = 0

Should any combination other than the one shown above exist, the DCD must take proper action. Transfer failure mechanisms are indicated in the Device Error Matrix.

In addition to checking the status bit the DCD must read the Transfer Bytes field to determine the actual bytes transferred. When a transfer is complete, the Total Bytes transferred is by decremented by the actual bytes transferred. For Transmit packets, a packet is only complete after the actual bytes reaches zero, but for receive packets, the host may send fewer bytes in the transfer according the USB variable length packet protocol.

### 21.8.5.5 Flushing/De-Priming an Endpoint

It is necessary for the DCD to flush to de-prime one more endpoints on a USB device reset or during a broken control transfer. There may also be application specific requirements to stop transfers in progress. The following procedure can be used by the DCD to stop a transfer in progress:

1. Write a '1' to the corresponding bit(s) in ENDPTFLUSH.
2. Wait until all bits in ENDPTFLUSH are ‘0.’
3. Software note: this operation may take a large amount of time depending on the USB bus activity. It is not desirable to have this wait loop within an interrupt service routine.
4. Read ENDPTSTATUS to ensure that for all endpoints commanded to be flushed, that the corresponding bits are now ‘0.’ If the corresponding bits are ‘1’ after step #2 has finished, then the flush failed as described in the following:

   Explanation: In very rare cases, a packet is in progress to the particular endpoint when commanded flush using ENDPTFLUSH. A safeguard is in place to refuse the flush to ensure that the packet in progress completes successfully. The DCD may need to repeatedly flush any endpoints that fail to flush be repeating steps 1–3 until each endpoint is successfully flushed.

### 21.8.5.6 Device Error Matrix

Table 21-90 summarizes packet errors that are not automatically handled by the USB controller:

**Table 21-90. Device Error Matrix**

| Error | Direction | Packet Type | Data Buffer Error Bit | Transaction Error Bit |
|---|---|---|---|---|
| Overflow ** | RX | Any | 1 | 0 |
| ISO Packet Error | RX | ISO | 0 | 1 |
| ISO Fulfillment Error | Both | ISO | 0 | 1 |

Notice that the device controller handles all errors on Bulk/Control/Interrupt Endpoints except for a data buffer overflow. However, for ISO endpoints, errors packets are not retried and errors are tagged as indicated.

**Table 21-91. Error Descriptions**

| | |
|---|---|
| **Overflow** | Number of bytes received exceeded max. packet size or total buffer length.<br>** This error will also set the Halt bit in the dQH and if there are dTDs remaining in the linked list for the endpoint, then those will not be executed. |
| **ISO Packet Error** | CRC Error on received ISO packet. Contents not guaranteed to be correct. |
| **ISO Fulfillment Error** | Host failed to complete the number of packets defined in the dQH mult field within the given (micro)frame. For scheduled data delivery the DCD may need to readjust the data queue because a fulfillment error will cause Device Controller to cease data transfers on the pipe for one (micro)frame. During the ‘dead’ (micro)frame, the Device Controller reports error on the pipe and primes for the following frame. |

## 21.8.6 Servicing Interrupts

The interrupt service routine must consider that there are high-frequency, low-frequency operations, and error operations and order accordingly.

### 21.8.6.1 High-Frequency Interrupts

High frequency interrupts in particular should be handed in the order below. The most important of these is listed first because the DCD must acknowledge a setup buffer in the timeliest manner possible.

**Table 21-92. Interrupt Handling Order**

| Execution Order | Interrupt | Action |
|---|---|---|
| 1a | USB Interrupt[1]<br>ENDPTSETUPSTATUS | Copy contents of setup buffer and acknowledge setup packet (as indicated in section Managing Queue Heads). Process setup packet according to USB 2.0 Chapter 9 or application specific protocol. |
| 1b | USB Interrupt<br>ENDPTCOMPLETE | Handle completion of dTD as indicated in section Managing Queue Heads. |
| 2 | SOF Interrupt | Action as deemed necessary by application. This interrupt may not have a use in all applications. |

[1] It is likely that multiple interrupts to stack up on any call to the Interrupt Service Routine AND during the Interrupt Service Routine.

### 21.8.6.2 Low-Frequency Interrupts

The low frequency events include the following interrupts. These interrupt can be handled in any order since they don't occur often in comparison to the high-frequency interrupts.

Table 21-93. Low Frequency Interrupt Events

| Interrupt | Action |
|---|---|
| Port Change | Change software state information. |
| Sleep Enable (Suspend) | Change software state information. Low power handling as necessary. |
| Reset Received | Change software state information. Abort pending transfers. |

### 21.8.6.3 Error Interrupts

Error interrupts will be least frequent and should be placed last in the interrupt service routine.

Table 21-94. Error Interrupt Events

| Interrupt | Action |
|---|---|
| USB Error Interrupt | This error is redundant because it combines USB Interrupt and an error status in the dTD. The DCD will more aptly handle packet-level errors by checking dTD status field upon receipt of USB Interrupt (w/ ENDPTCOMPLETE). |
| System Error | Unrecoverable error. Immediate Reset of core; free transfers buffers in progress and restart the DCD. |

## 21.9 Deviations from the EHCI Specifications

The host mode operation of the USB DR module is nearly EHCI-compatible with few minor differences. For the most part, the module conforms to the data structures and operations described in Section 3, “Data Structures,” and Section 4, “Operational Model,” in the EHCI specification. The particulars of the deviations occur in the following areas:

- Embedded transaction translator—Allows direct attachment of FS and LS devices in host mode without the need for a companion controller.
- Device operation—In host mode, the device operational registers are generally disabled and thus device mode is mostly transparent when in host mode. However, there are a couple exceptions documented in the following sections.
- Embedded design interface—The module does not have a PCI interface and therefore the PCI configuration registers described in the EHCI specification are not applicable.

### 21.9.1 Embedded Transaction Translator Function

The USB module supports directly connected full and low speed devices without requiring a companion controller by including the capabilities of a USB 2.0 high speed hub transaction translator. Although there is no separate transaction translator block in the system, the transaction translator function normally associated with a high speed hub has been implemented within the DMA and Protocol engine blocks. The embedded transaction translator function is an extension to EHCI interface, but makes use of the standard data structures and operational models that exist in the EHCI specification to support full and low speed devices.

#### 21.9.1.1 Capability Registers

The following additions have been added to the capability registers to support the embedded transaction translator Function:

- N_TT added to HSCPARAMS - Host Controller Structural Parameters
- N_PTT added to HSCPARAMS - Host Controller Structural Parameters

See Section 21.3.1.3, "Host Controller Structural Parameters (HCSPARAMS)," for usage information.

#### 21.9.1.2 Operational Registers

The following additions have been added to the operational registers to support the embedded TT:

- ASYNCTTSTS is a new register.
- Addition of two-bit Port Speed (PSPD) to the PORTSC register.

#### 21.9.1.3 Discovery

In a standard EHCI controller design, the EHCI host controller driver detects a Full speed (FS) or Low speed (LS) device by noting if the port enable bit is set after the port reset operation. The port enable will only be set in a standard EHCI controller implementation after the port reset operation and when the host and device negotiate a High-Speed connection (that is, Chirp completes successfully).

The module will always set the port enable after the port reset operation regardless of the result of the host device chirp result and the resulting port speed will be indicated by the PSPD field in PORTSC. Therefore, the standard EHCI host controller driver requires an alteration to handle directly connected Full and Low speed devices or hubs. The change is a fundamental one in that is summarized in Table 21-95.

**Table 21-95. Functional Differences Between EHCI and EHCI with Embedded TT**

| Standard EHCI | EHCI with Embedded Transaction Translator |
|---|---|
| After port enable bit is set following a connection and reset sequence, the device/hub is assumed to be HS. | After port enable bit is set following a connection and reset sequence, the device/hub speed is noted from PORTSC. |
| FS and LS devices are assumed to be downstream from a HS hub thus, all port-level control is performed through the Hub Class to the nearest Hub. | FS and LS device can be either downstream from a HS hub or directly attached. When the FS/LS device is downstream from a HS hub, then port-level control is done using the Hub Class through the nearest Hub. When a FS/LS device is directly attached, then port-level control is accomplished using PORTSC. |
| FS and LS devices are assumed to be downstream from a HS hub with HubAddr=X. [where HubAddr > 0 and HubAddr is the address of the Hub where the bus transitions from HS to FS/LS (that is, Split target hub)] | FS and LS device can be either downstream from a HS hub with HubAddr = X [HubAddr > 0] or directly attached [where HubAddr = 0 and HubAddr is the address of the Root Hub where the bus transitions from HS to FS/LS (that is, Split target hub is the root hub)] |

#### 21.9.1.4 Data Structures

The same data structures used for FS/LS transactions though a HS hub are also used for transactions through the Root Hub. Here it is demonstrated how the Hub Address and Endpoint Speed fields should be set for directly attached FS/LS devices and hubs:

1. QH (for direct attach FS/LS)—Async. (Bulk/Control Endpoints) Periodic (Interrupt)

- Hub Address = 0
- Transactions to direct attached device/hub.
  - QH.EPS = Port Speed
- Transactions to a device downstream from direct attached FS hub.
  - QH.EPS = Downstream Device Speed

**NOTE**

> When QH.EPS = 01 (LS) and PORTSC[PSPD] = 00 (FS), a LS-pre-pid will be sent before the transmitting LS traffic.

Maximum Packet Size must be less than or equal 64 or undefined behavior may result.

2. siTD (for direct attach FS)—Periodic (ISO Endpoint)

- All FS ISO transactions:
  - Hub Address = 0
  - siTD.EPS = 00 (full speed)

Maximum Packet Size must less than or equal to 1023 or undefined behavior may result.

#### 21.9.1.5 Operational Model

The operational models are well defined for the behavior of the transaction translator (see *Universal Serial Bus Revision 2.0 Specification*) and for the EHCI controller moving packets between system memory and a USB-HS hub. Since the embedded transaction translator exists within the USB module there is no physical bus between EHCI host controller driver and the USB FS/LS bus. These sections will briefly discuss the operational model for how the EHCI and transaction translator operational models are combined without the physical bus between. The following sections assume the reader is familiar with both the EHCI and USB 2.0 transaction translator operational models.

##### 21.9.1.5.1 Microframe Pipeline

The EHCI operational model uses the concept of H-frames and B-frames to describe the pipeline between the Host (H) and the Bus (B). The embedded transaction translator shall use the same pipeline algorithms specified in the *Universal Serial Bus Revision 2.0 Specification* for a Hub-based transaction translator.

All periodic transfers always begin at B-frame 0 (after SOF) and continue until the stored periodic transfers are complete. As an example of the microframe pipeline implemented in the embedded transaction translator, all periodic transfers that are tagged in EHCI to execute in H-frame 0 will be ready to execute on the bus in B-frame 0.

It is important to note that when programming the S-mask and C-masks in the EHCI data structures to schedule periodic transfers for the embedded transaction translator, the EHCI host controller driver must follow the same rules specified in EHCI for programming the S-mask and C-mask for downstream Hub-based transaction translators.

Once periodic transfers are exhausted, any stored asynchronous transfer will be moved. Asynchronous transfers are opportunistic in that they shall execute whenever possible and their operation is not tied to

H-frame and B-frame boundaries with the exception that an asynchronous transfer can not babble through the SOF (start of B-frame 0).

#### 21.9.1.5.2 Split State Machines

The start and complete split operational model differs from EHCI slightly because there is no bus medium between the EHCI controller and the embedded transaction translator. Where a start or complete-split operation would occur by requesting the split to the HS hub, the start/complete split operation is simple an internal operation to the embedded transaction translator. Table 21-96 summarizes the conditions where handshakes are emulated from internal state instead of actual handshakes to HS split bus traffic.

**Table 21-96. Emulated Handshakes**

| Condition | Emulate TT Response |
|---|---|
| **Start-Split:** All asynchronous buffers full | NAK |
| **Start-Split:** All periodic buffers full | ERR |
| **Start-Split:** Success for start of Async. Transaction | ACK |
| **Start-Split:** Start Periodic Transaction | No Handshake (Ok) |
| **Complete-Split:** Failed to find transaction in queue | Bus Time Out |
| **Complete-Split:** Transaction in Queue is Busy | NYET |
| **Complete-Split:** Transaction in Queue is Complete | [Actual Handshake from FS/LS device] |

#### 21.9.1.5.3 Asynchronous Transaction Scheduling and Buffer Management

The following *Universal Serial Bus Revision 2.0 Specification* items are implemented in the embedded transaction translator:

- USB 2.0–11.17.3
  - Sequencing is provided and a packet length estimator ensures no full-speed/low-speed packet babbles into SOF time.
- USB 2.0–11.17.4
  - • Transaction tracking for 2 data pipes.
- USB 2.0–11.17.5
  - • Clear_TT_Buffer capability provided

#### 21.9.1.5.4 Periodic Transaction Scheduling and Buffer Management

The following *Universal Serial Bus Revision 2.0 Specification* items are implemented in the embedded transaction translator:

- USB 2.0–11.18.6.[1-2]
  - Abort of pending start-splits
    - EOF (and not started in microframes 6)
    - Idle for more than 4 microframes
  - Abort of pending complete-splits

- EOF
- Idle for more than 4 microframes

**NOTE**

There is no data schedule mechanism for these transactions other than the microframe pipeline. The embedded TT assumes the number of packets scheduled in a frame does not exceed the frame duration (1 msec) or else undefined behavior may result.

#### 21.9.1.5.5 Multiple Transaction Translators

The maximum number of embedded transaction translators that is currently supported is one as indicated by the N_TT field in the HCSPARAMS register. See Section 21.3.1.3, “Host Controller Structural Parameters (HCSPARAMS),” for more information.

### 21.9.2 Device Operation

The co-existence of a device operational controller within the USB DR module has little effect on EHCI compatibility for host operation except as noted in this section.

### 21.9.3 Non-Zero Fields the Register File

Some of the reserved fields and reserved addresses in the capability registers and operational registers have use in device mode, the following must be adhered to:

- Write operations to all EHCI reserved fields (some of which are device fields in the USB module) in the operation registers should always be written to zero. This is an EHCI requirement of the device controller driver that must be adhered to.
- Read operations by the module must properly mask EHCI reserved fields (some of which are device fields in the USB module registers).

### 21.9.4 SOF Interrupt

The SOF interrupt is a free running 125 µsec interrupt for host mode. EHCI does not specify this interrupt, but it has been added for convenience and as a potential software time base. Note that the free running interrupt is shared with the device-mode start-of-frame interrupt. See Section 21.3.2.2, “USB Status Register (USBSTS),” and Section 21.3.2.3, “USB Interrupt Enable Register (USBINTR),” for more information.

### 21.9.5 Embedded Design

This is an Embedded USB Host Controller as defined by the EHCI specification and thus does not implement the PCI configuration registers.

#### 21.9.5.1 Frame Adjust Register

Given that the optional PCI configuration registers are not included in this implementation, there is no corresponding bit level timing adjustments like those provided by the Frame Adjust register in the PCI configuration registers. Starts of microframes are timed precisely to 125 µsec using the transceiver clock as a reference clock. That is, 60 MHz transceiver clock for 8-bit physical interfaces and full-speed serial interfaces or 30 MHz transceiver clock for 16-bit physical interfaces.

### 21.9.6 Miscellaneous Variations from EHCI

#### 21.9.6.1 Programmable Physical Interface Behavior

The modules support multiple physical interfaces which can operate in different modes when the module is configured with the software programmable Physical Interface Modes. The control bits for selecting the PHY operating mode have been added to the PORTSC register providing a capability that is not defined by the EHCI specification.

#### 21.9.6.2 Discovery

##### 21.9.6.2.1 Port Reset

The port connect methods specified by EHCI require setting the port reset bit in the register for a duration of 10 msec. Due to the complexity required to support the attachment of devices that are not high speed there are counter already present in the design that can count the 10 msec reset pulse to alleviate the requirement of the software to measure this duration. Therefore, the basic connection is then summarized as the following:

- [Port Change Interrupt] Port connect change occurs to notify the host controller driver that a device has attached.
- Software shall write a ‘1’ to the reset the device.
- Software shall write a ‘0’ to the reset the device after 10 msec.
  - This step, which is necessary in a standard EHCI design, may be omitted with this implementation. Should the EHCI host controller driver attempt to write a ‘0’ to the reset bit while a reset is in progress the write will simple be ignored and the reset will continue until completion.
- [Port Change Interrupt] Port enable change occurs to notify the host controller that the device in now operational and at this point the port speed has been determined.

##### 21.9.6.2.2 Port Speed Detection

After the port change interrupt indicates that a port is enabled, the EHCI stack should determine the port speed. Unlike the EHCI implementation which will re-assign the port owner for any device that does not connect at High-Speed, this host controller supports direct attach of non-HS devices. Therefore, the following differences are important regarding port speed detection:

- Port owner is read-only and always reads 0.

- A 2-bit port speed indicator has been added to PORTSC to provide the current operating speed of the port to the host controller driver.
- A 1-bit high-speed indicator has been added to PORTSC to signify that the port is in HS vs. FS/LS

## 21.10 Timing Diagrams

This section contains diagrams showing the basic operation of the ULPI interface. For a more detailed description refer to the ULPI Specifications.

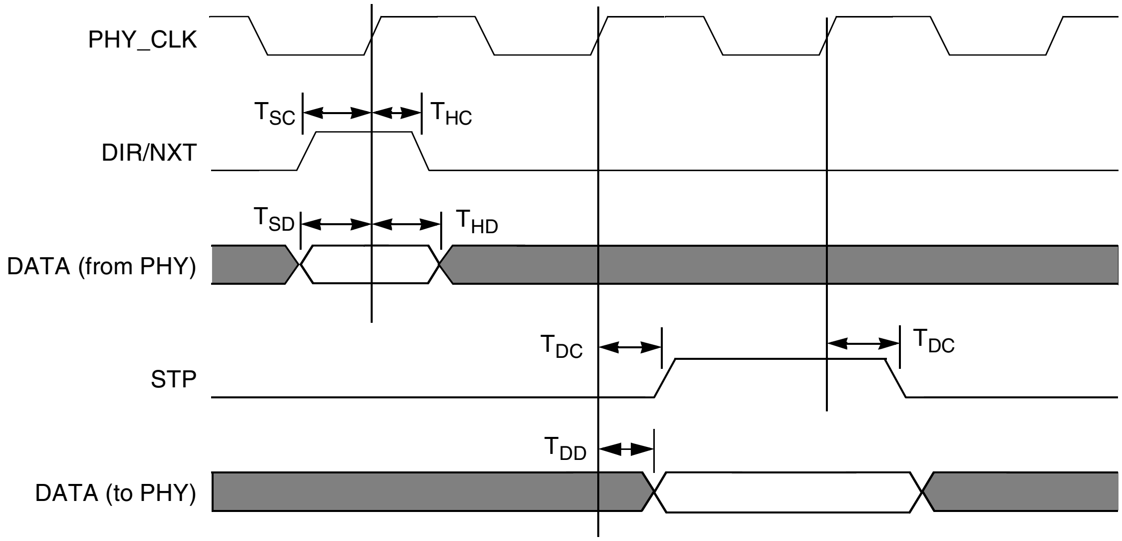


**Figure 21-65. ULPI Timing**

**Table 21-97. ULPI Timing**

| Parameter | Symbol | Min | Max | Units |
|---|---|---|---|---|
| Control signal setup time | $T_{SC}$ | — | 4 | ns |
| Data setup time | $T_{SD}$ | — | 4 | ns |
| Control signal hold time | $T_{HC}$ | 0 | — | ns |
| Data hold time | $T_{HD}$ | 0 | — | ns |
| Control output delay | $T_{DC}$ | 2 | 7 | ns |
| Data output delay | $T_{DD}$ | 2 | 7 | ns |

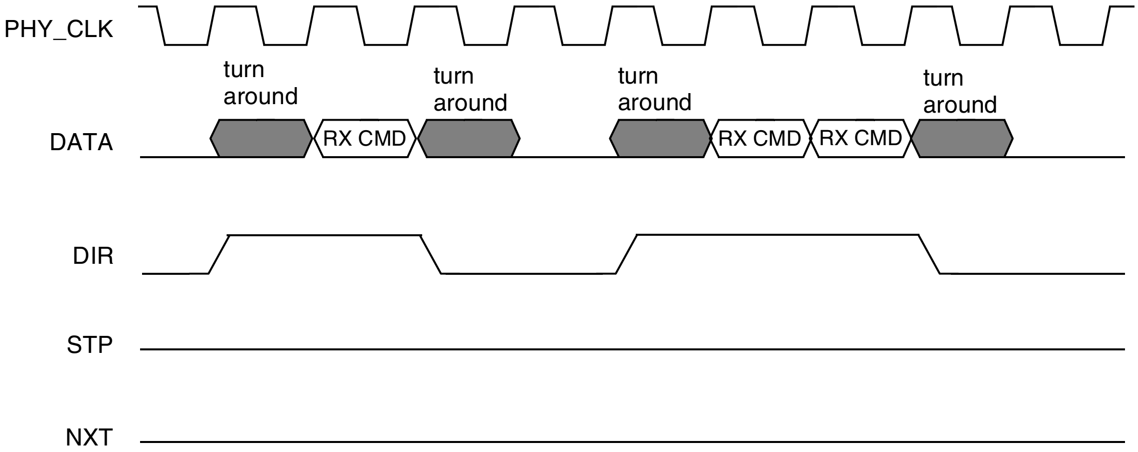


Figure 21-66. Sending of RX CMD

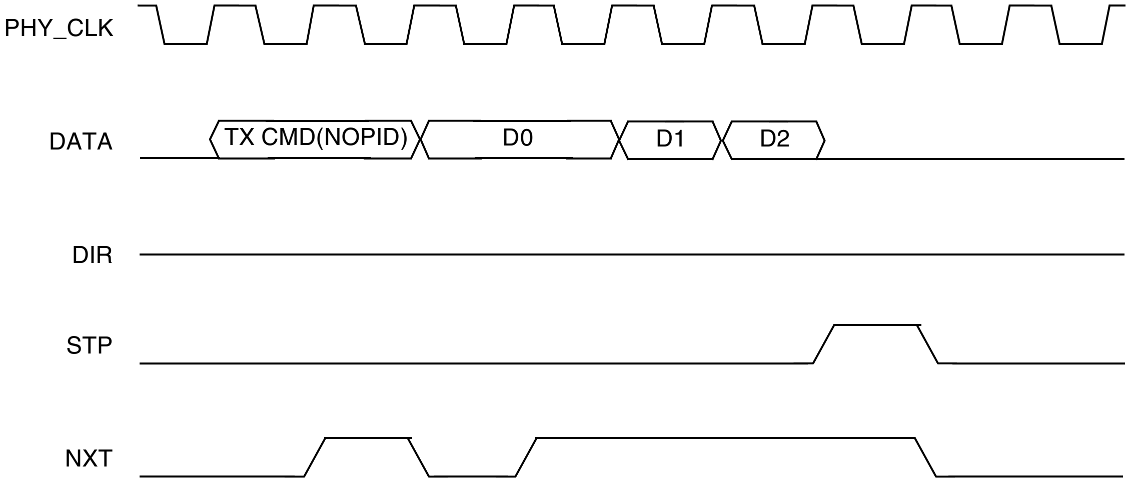


Figure 21-67. ULPI Data Transmit (NOPID)

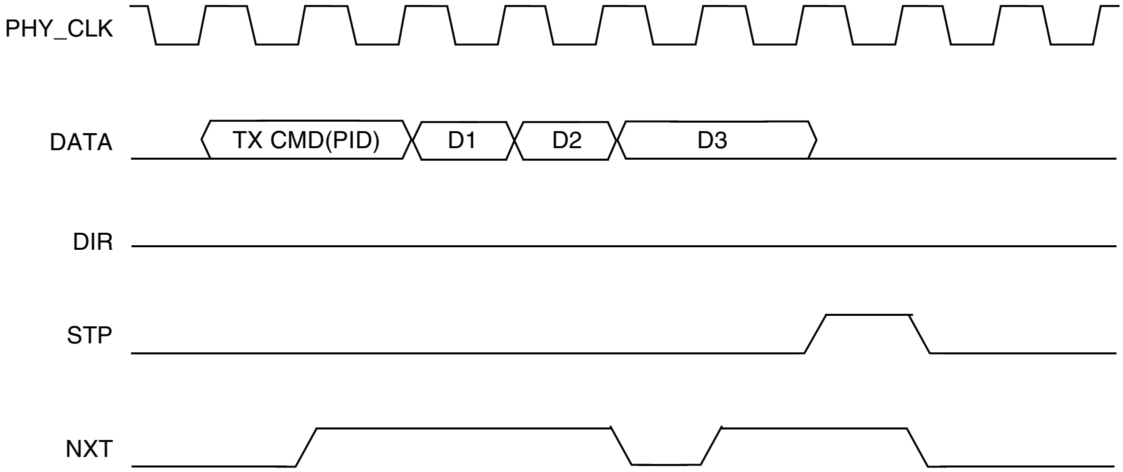


Figure 21-68. ULPI Data Transmit (PID)

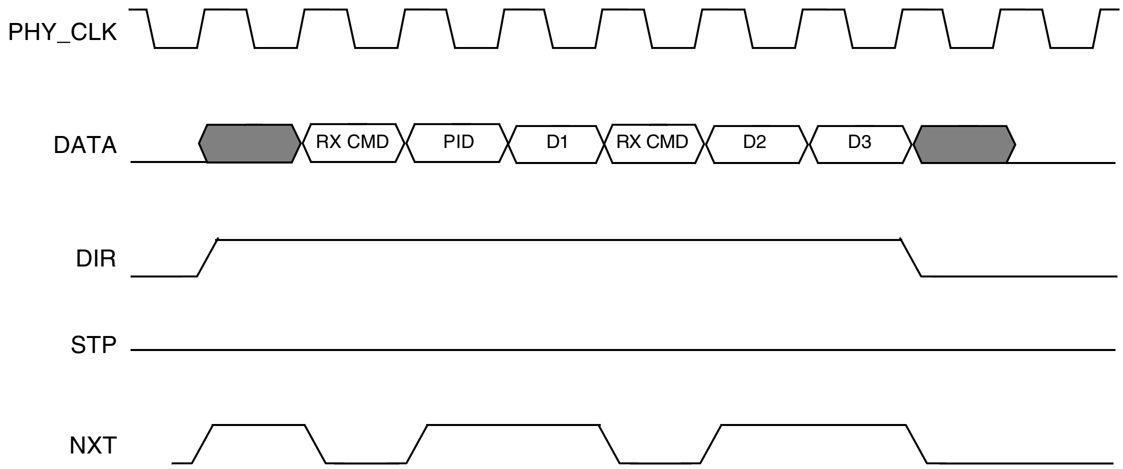


Figure 21-69. ULPI Data Receive

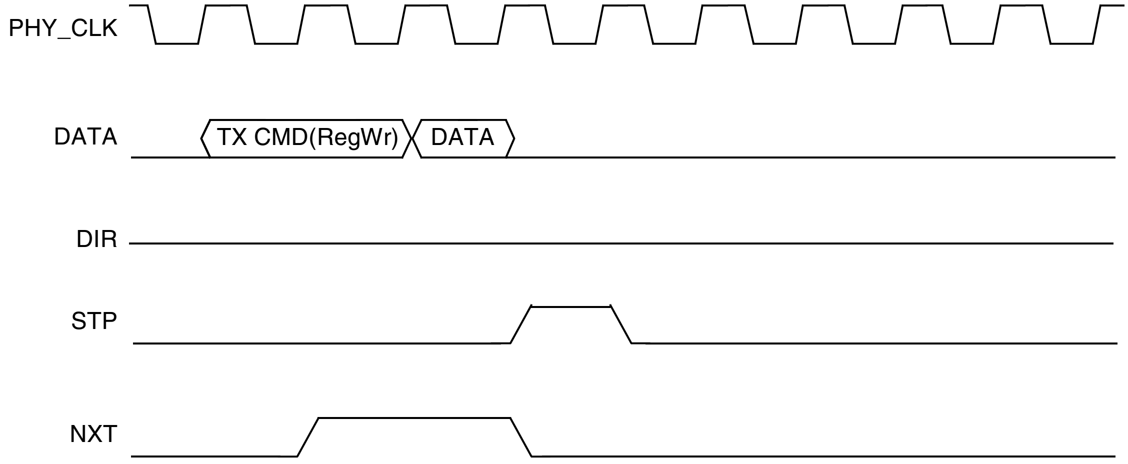


Figure 21-70. ULPI Register Write

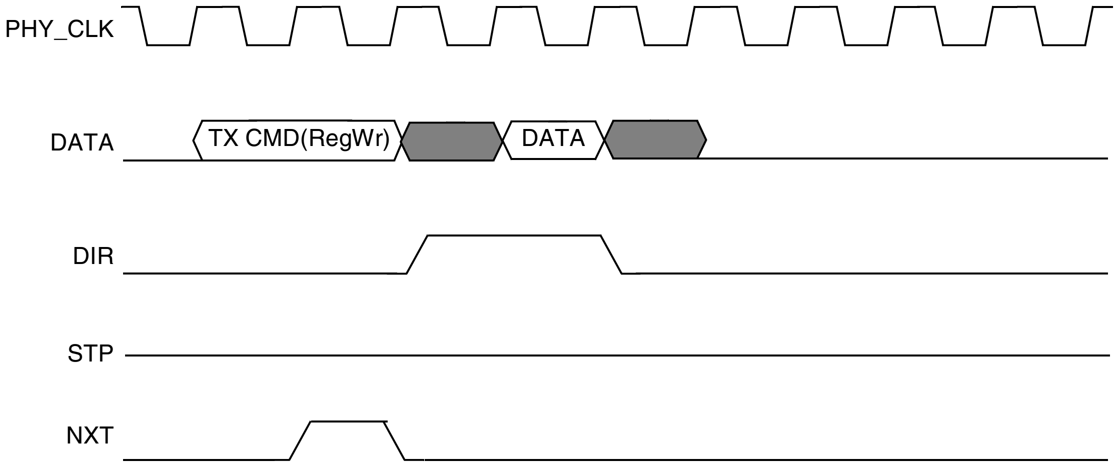


Figure 21-71. ULPI Register Read

# Chapter 22
# General Purpose I/O (GPIO)

## 22.1 Introduction

This chapter describes the general-purpose I/O module, including pin descriptions, register settings, and interrupt capabilities. Figure 22-1 shows the block diagram of the GPIO module.

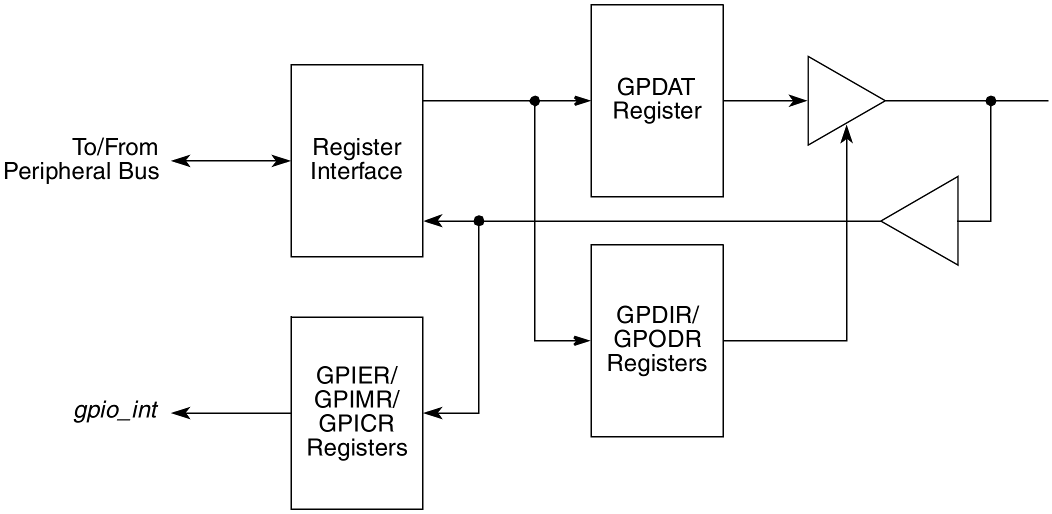


**Figure 22-1. GPIO Module Block Diagram**

### 22.1.1 Overview

The GPIO module supports  general-purpose I/O ports.Each port can be configured as an input or as an output. If a port is configured as an input, it can optionally generate an interrupt on detection of a change. If a port is configured as an output, it can be individually configured as an open-drain or a fully active output.

### 22.1.2 Features

The GPIO unit implements the following features:

-  input/output ports
- All signals are configured as inputs when the device comes out of reset and also when $\overline{\text{HRESET}}$ is asserted.
- Open-drain capability on all ports
- All ports can optionally generate an interrupt upon changing their state.

## 22.2 External Signal Description

The following section provides information about GPIO signals.

### 22.2.1 Signals Overview

Table 22-1 provides detailed descriptions of the external GPIO signals.

**Table 22-1. IPIC External Signals—Detailed Signal Descriptions**

| Signal | I/O | Description | |
|---|---|---|---|
| GPIO[0:15] | I/O | General purpose I/O. Each signal can be set individually to act as input or output, according to application needs. | |
| | | **State Meaning** | Asserted/Negated—Defined per application. |
| | | **Timing** | Assertion/Negation—Inputs can be asserted completely asynchronously.<br>Outputs are asynchronous to any externally visible clock |

## 22.3 Memory Map/Register Definition

The GPIO has programmable registers that occupy memory-mapped space. Note that reading undefined portions of the memory map returns all zeros and writing has no effect.

All GPIO registers are 32 bits wide and are located on 32-bit address boundaries.

Table 22-2 shows the memory map of GPIO.

**Table 22-2. GPIO Register Address Map**

| Offset | Register | Access | Reset Value | Section/Page |
|---|---|---|---|---|
| 0xC00 | GPIO direction register (GPDIR) | R/W | 0x0000_0000 | 22.3.1/22-2 |
| 0xC04 | GPIO open drain register (GPODR) | R/W | 0x0000_0000 | 22.3.2/22-3 |
| 0xC08 | GPIO data register (GPDAT) | R/W | 0x0000_0000 | 22.3.3/22-3 |
| 0xC0C | GPIO interrupt event register (GPIER) | w1c | Undefined | 22.3.4/22-4 |
| 0xC10 | GPIO interrupt mask register (GPIMR) | R/W | 0x0000_0000 | 22.3.5/22-4 |
| 0xC14 | GPIO external interrupt control register (GPICR) | R/W | 0x0000_0000 | 22.3.6/22-5 |

### 22.3.1 GPIO Direction Register (GPDIR)

The GPIO direction registers (GPDIR), shown in Figure 22-2, defines the direction of the individual ports.

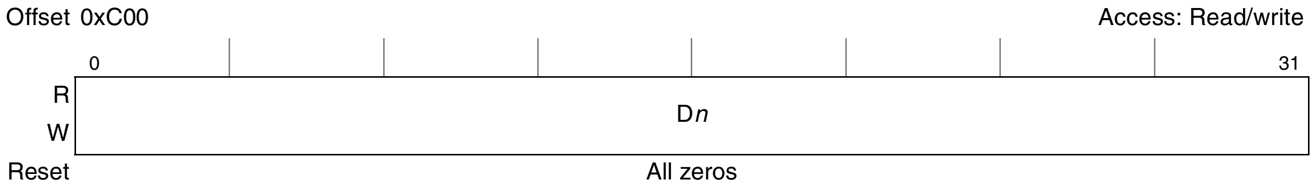


**Figure 22-2. GPIO Direction Register (GPDIR)**

Table 22-3 defines the bit fields of GPDIR.

**Table 22-3. GPDIR Bit Settings**

| Bits | Name | Description |
|---|---|---|
| 0–31 | D*n* | Direction. Indicates whether a signal is used as an input or an output. Bits D0–D15 correspond to signals GPIO[0:15]. Bits D16–D31 are unused.<br>0 The corresponding signal is an input.<br>1 The corresponding signal is an output. |

## 22.3.2 GPIO Open Drain Register (GPODR)

The GPIO open drain register (GPODR), shown in Figure 22-3, defines the way individual ports drive their output.

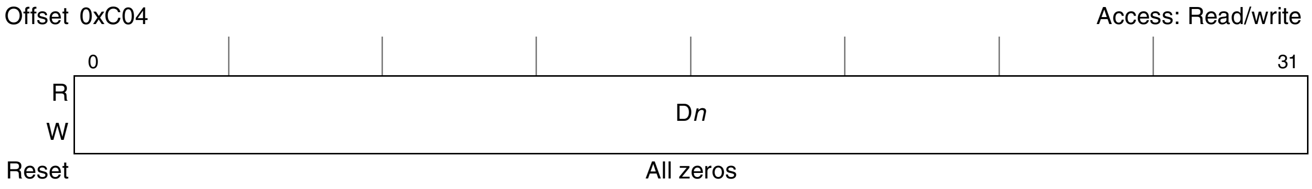


**Figure 22-3. GPIO Open Drain Register (GPODR)**

Table 22-4 defines the bit fields of GPODR.

**Table 22-4. GPODR Bit Settings**

| Bits | Name | Description |
|---|---|---|
| 0–31 | D*n* | Open-drain configuration. Indicates whether a signal is actively driven as an output or an open-drain driver. This register has no effect on signals programmed as inputs in the corresponding GPDIR. Bits D0–D15 correspond to signals GPIO[0:15]. Bits D16–D31 are unused.<br>0 The I/O signal is actively driven as an output.<br>1 The I/O signal is an open-drain driver. As an output, the signal is driven active-low, otherwise it is three-stated. |

## 22.3.3 GPIO Data Register (GPDAT)

The GPIO data register (GPDAT), shown in Figure 22-4, carries the data in/out for the individual ports.

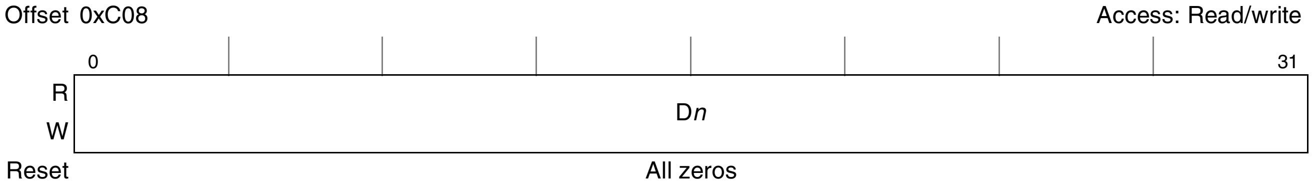


**Figure 22-4. GPIO Data Register (GPDAT)**

Table 22-5 defines the bit fields of GPDAT.

Table 22-5. GP*n*DAT Bit Settings

| Bits | Name | Description |
|---|---|---|
| 0–31 | D*n* | Data. Write data is latched and presented on external signals if GPDIR has configured the port as an output. Read operation always returns the data at the signal. Bits D0–D15 correspond to signals GPIO[0:15]. Bits D16–D31 are unused. |

### 22.3.4 GPIO Interrupt Event Register (GPIER)

The GPIO interrupt event register (GPIER), shown in Figure 22-5, carries information of the events that caused an interrupt. Each bit in GPIER, corresponds to an interrupt source. GPIER bits are cleared by writing ones. However, writing zero has no effect.

Offset 0xC0C Access: w1c

| | 0 — 31 |
|---|---|
| R | D*n* |
| W | |
| Reset | Undefined (the user should write 1s to clear before using) |

Figure 22-5. GPIO Interrupt Event Register (GPIER)

Table 22-6 defines the bit fields of GPIER.

Table 22-6. GPIER Bit Settings

| Bits | Name | Description |
|---|---|---|
| 0–31 | D*n* | Interrupt events. Indicates whether an interrupt event occurred on the corresponding GPIO signal. Bits D0–D15 correspond to signals GPIO[0:15]. Bits D16–D31 are unused.<br>0 No interrupt event occurred on the corresponding GPIO signal.<br>1 Interrupt event occurred on the corresponding GPIO signal. |

### 22.3.5 GPIO Interrupt Mask Register (GPIMR)

The GPIO interrupt mask register (GPIMR), shown in Figure 22-6, defines the interrupt masking for the individual ports. When a masked interrupt request occurs, the corresponding GPIER bit is set, regardless of the GPIMR state. When one or more non-masked interrupt events occur, the GPIO module issues an interrupt to the on chip interrupt controller.

Offset 0xC10 Access: Read/write

| | 0 — 31 |
|---|---|
| R | D*n* |
| W | |
| Reset | All zeros |

Figure 22-6. GPIO Interrupt Mask Register (GPIMR)

Table 22-7 defines the bit fields of GPIMR.

Table 22-7. GPIMR Bit Settings

| Bits | Name | Description |
|---|---|---|
| 0–31 | D*n* | Interrupt mask. Indicates whether an interrupt event is masked or not masked. Bits D0–D15 correspond to signals GPIO[0:15]. Bits D16–D31 are unused.<br>0 The input interrupt signal is masked (disabled).<br>1 The input interrupt signal is not masked (enabled). |

### 22.3.6 GPIO Interrupt Control Register (GPICR)

The GPIO interrupt control register (GPICR), shown in Figure 22-7, determines whether the corresponding port line asserts an interrupt request on either a high-to-low change or any change on the state of the signal.

Offset 0xC14 Access: Read/write

| | 0 … 31 |
|---|---|
| R<br>W | D*n* |
| Reset | All zeros |

Figure 22-7. GPIO Interrupt Control Register (GPICR)

Table 22-8 defines the bit fields of GPICR.

Table 22-8. GPICR Bit Settings

| Bits | Name | Description |
|---|---|---|
| 0–31 | D*n* | Edge detection mode. Bits D0–D15 correspond to signals GPIO[0:15]. Bits D16–D31 are unused. The corresponding port line asserts an interrupt request according to the following:<br>0 Any change on the state of the port generates an interrupt request.<br>1 High-to-low change on the port generates an interrupt request. |

# Part IV
# Global Functions and Debug

Part IV defines other global blocks of the MPC8536E. The following chapters are included:

- Chapter 23, “Global Utilities,” defines the global utilities of the MPC8536E. These include power management, I/O device enabling, power-on-reset (POR) configuration monitoring, general-purpose I/O signal use, and multiplexing for the interrupt and local bus chip select signals.
- Chapter 24, “Device Performance Monitor,” describes the performance monitor of the MPC8536E. Note that the MPC8536E performance monitor is similar to but separate from the performance monitor implemented on the e500v2 core.
- Chapter 25, “Debug Features and Watchpoint Facility,” describes the debug features and watchpoint monitor of the MPC8536E.

# Chapter 23
# Global Utilities

This chapter describes the global utilities of the MPC8536E. It provides signal descriptions, register descriptions, and a functional description of these utilities.

## 23.1 Overview

The global utilities block controls power management, I/O device enabling, power-on-reset (POR) configuration monitoring, general-purpose I/O signal configuration, alternate function selection for multiplexed signals, and clock control.

## 23.2 Global Utilities Features

This section provides an overview of global utilities features.

### 23.2.1 Power Management and Block Disables

The following features affect the device's overall power consumption:

- Dynamic power management mode
- Software-controlled power management (doze, nap, sleep)
- Externally controlled power management (doze, sleep, deep sleep)
- Static power management (I/O block disables)

### 23.2.2 Accessing Current POR Configuration Settings

The POR configuration values of all device parameters sampled from pins at reset are available through memory-mapped registers in the global utilities block.

### 23.2.3 Signal Multiplexing

Many of the signals serve multiple functions that can be selected by configuration registers in the global utilities block. See Section 23.4.1.9, "Alternate Function Signal Multiplex Control Register (PMUXCR)," for more information.

### 23.2.4 Clock Control

The global utilities block also selects the internal clock signal driven on CLK_OUT.

## 23.3 External Signal Description

The following subsections provide information about signals that serve as global utilities.

### 23.3.1 Signals Overview

Table 23-1 summarizes the external signals used by the global utilities block.

**Table 23-1. External Signal Summary**

| Signal Name | I/O | Description | Reference (Section/page) |
|---|---|---|---|
| ASLEEP | O | Signals that the device has reached a sleep state. | 23.5.1.6.3/23-50 |
| CKSTP_IN | I | Checkstop input | Table 23-2 on page 23-2 |
| CKSTP_OUT | O | Checkstop output. | Table 23-2 on page 23-2 |
| CLK_OUT | O | Clock out. Selected by CLKOCR values. | 23.4.1.25/23-33 |
| POWER_EN | O | Indication to turn the power on and off. | 23.5.1.16.2/23-61 |
| POWER_OK | I | Indication that the power has returned to specified level after a wakeup event occurs. | 23.5.1.16.1/23-60 |

### 23.3.2 Detailed Signal Descriptions

Table 23-2 describes signals in the global utilities block in detail.

**Table 23-2. Detailed Signal Descriptions**

| Signal | I/O | Description | |
|---|---|---|---|
| ASLEEP | O | Asleep. See Section 23.5.1.6.3, “Sleep Mode.” After negation of HRESET, ASLEEP is asserted until the device completes its power-on reset sequence and reaches its ready state. | |
| | | **State Meaning** | Asserted—Indicates that the device is either still in its power-on reset sequence or it has reached a sleep state after a power-down command is issued by software.<br>Negated—The device is not in sleep mode. (It has either awakened from a power-down state, or has completed the POR sequence.) |
| | | **Timing** | Assertion—May occur at any time; may be asserted asynchronously to the input clocks.<br>Negation—Negates synchronously with SYSCLK when leaving power-on sequence; otherwise negation is asynchronous. |
| CKSTP_IN | I | Checkstop in | |
| | | **State Meaning** | Asserted—Indicates that the e500 core must enter a hard stop condition. All e500 clocks are turned off. CKSTP_OUT is asserted. The rest of MPC8536E device logic, including memory controllers, internal memories and registers, and I/O interfaces, remains functional.<br>Negated—Indicates that normal operation should proceed. |
| | | **Timing** | Assertion—May occur at any time; may be asserted asynchronously to the input clocks.<br>Negation—Must remain asserted until the MPC8536E is reset with assertion of HRESET. |

**Table 23-2. Detailed Signal Descriptions (continued)**

| Signal | I/O | Description | |
|---|---|---|---|
| $\overline{\text{CKSTP\_OUT}}$ | O | Checkstop out | |
| | | **State Meaning** | Asserted—Indicates that the e500 core of the MPC8536E is in a checkstop state. The rest of the MPC8536E logic remains functional unless.<br>Negated—Indicates normal operation. After $\overline{\text{CKSTP\_OUT}}$ has been asserted, it is negated after the next negation (low-to-high transition) of $\overline{\text{HRESET}}$. |
| | | **Timing** | Assertion—May occur at any time; may be asserted asynchronously to the input clocks.<br>Negation—Must remain asserted until the device has been reset with a hard reset. |
| CLK_OUT | O | Clock out. Reflects clock signal selected by CLKOCR (see Section 23.4.1.25, “Clock Out Control Register (CLKOCR)”). | |
| | | **State Meaning** | Asserted—If CLKOCR[ENB] = 1, clock signal selected by CLKOCR[CLK_SEL] is driven.<br>High impedance—If CLKOCR[ENB] = 0. |
| | | **Timing** | Assertion/Negation—Depends on the value of CLKOCR[CLK_SEL]. |
| POWER_EN | O | Power enable | |
| | | **State Meaning** | Asserted—Indicates to the external power regulator to toggle the power switch to on mode.<br>Negated—Indicates to the external power regulator to toggle the power switch to off mode. Reset value is 1. |
| | | **Timing** | Assertion—May occur only when a wakeup event occurs.<br>Negation—No wakeup events occurs at the device. The timing of the signal is asynchronous; the signal is stable long enough so its possible to synchronize it. |
| POWER_OK | I | Power OK | |
| | | **State Meaning** | Asserted—Indicate that power level supplied by the external regulator is stable<br>Negated—Indicate that power supplied by the external regulator is off or not stable |
| | | **Timing** | Assertion—May occur when the power is stable while the power_en signal is asserted<br>Negation—Negates asynchronous with power down. The timing of the signal is asynchronous, the signal is stable long enough so its possible to synchronize it. |

## 23.4 Memory Map/Register Definition

Table 23-3 summarizes the global utilities registers and their addresses.

In this table and in the register figures and field descriptions, the following access definitions apply:

- Reserved fields are always ignored for the purposes of determining access type.
- R/W, R, and W (read/write, read only, and write only) indicate that all the non-reserved fields in a register have the same access type.
- w1c indicates that all of the non-reserved fields in a register are cleared by writing ones to them.
- Mixed indicates a combination of access types.
- Special is used when no other category applies. In this case the register figure and field description table should be read carefully.

Table 23-3. Global Utilities Module Memory Map

| Offset | Register | Access | Reset | Section/page |
|---|---|---|---|---|
| **Global Utilities Registers—Block Base Address 0xE_0000** | | | | |
| **Power-On Reset Configuration Values** | | | | |
| 0x000 | PORPLLSR—POR PLL Ratio Status Register | R | 0x*nnnn*_*nnnn* | 23.4.1.1/23-5 |
| 0x004 | PORBMSR—POR Boot Mode Status Register | R | 0x*nnnn*_0000 | 23.4.1.2/23-6 |
| 0x008 | PORIMPSCR—POR I/O Impedance Status And Control Register | Mixed | 0x000*n*_007F | 23.4.1.3/23-8 |
| 0x00C | PORDEVSR—POR Device Status Register | R | 0x*nnnn*_*nnn*0 | 23.4.1.4/23-9 |
| 0x010 | PORDBGMSR—POR Debug Mode Status Register | R | 0x0*n*00_0000 | 23.4.1.5/23-12 |
| 0x014 | PORDEVSR2—POR Device Status Register 2 | R | 0x*nn*00_001F | 23.4.1.6/23-13 |
| 0x020 | GPPORCR—General-purpose POR Configuration Register | R | 0x*nnnn*_*nnnn* | 23.4.1.7/23-13 |
| 0x030 | GENCFGR—General Configuration Register | R/W | 0x0000_0000 | 23.4.1.8/23-14 |
| **Signal Multiplexing Controls** | | | | |
| 0x060 | PMUXCR—Alternate Function Signal Multiplex Control | R/W | 0x0000_0000 | 23.4.1.9/23-14 |
| **Device Disables** | | | | |
| 0x070 | DEVDISR—Device Disable Control | R/W | 0x*nn*0*n*_0*n*0*n* | 23.4.1.10/23-16 |
| **Power Management Registers** | | | | |
| 0x07C | PMJCR—Power Management Jog Control Register | R/W | 0x00*nn*_0000 | 23.4.1.11/23-19 |
| 0x080 | POWMGTCSR—Power Management Status And Control Register | Mixed | 0x0000_0000 | 23.4.1.12/23-21 |
| 0x084 | PMRCCR—Power Management Reset Counters Configuration Register | R/W | 0x0C83_09D1 | 23.4.1.13/23-22 |
| 0x088 | PMPDCCR—Power Management Power Down Counters Configuration Register | R/W | 0x08D1_0000 | 23.4.1.14/23-24 |
| 0x08C | PMCDR—Power Management Clock Disable Register | R/W | 0x0000_0800 | 23.4.1.15/23-25 |
| **Interrupt and Reset Status and Control** | | | | |
| 0x090 | MCPSUMR—Machine Check Summary Register | w1c | 0x0000_0000 | 23.4.1.16/23-26 |
| 0x094 | RSTRSCR—Reset Request Status And Control Register | R | 0x0000_0000 | 23.4.1.17/23-27 |
| 0x098 | ECTRSTCR—Exception Reset Control Register | Mixed | 0x0000_0000 | 23.4.1.18/23-28 |
| 0x09C | RSTSR—Automatic Reset Status Register | Mixed | 0x0000_0000 | 23.4.1.19/23-28 |
| **Version Registers** | | | | |
| 0x0A0 | PVR—Processor version register | R | e500 processor version | 23.4.1.20/23-29 |
| 0x0A4 | SVR—System version register | R | MPC8536E system version | 23.4.1.21/23-30 |
| **Status Registers** | | | | |
| 0x0B0 | RSTCR—Reset control register | R/W | 0x0000_0000 | 23.4.1.22/23-30 |
| 0x0C0 | LBCVSELCR—LBC voltage select control register | R/W | 0x0000_0000 | 23.4.1.23/23-31 |

**Table 23-3. Global Utilities Module Memory Map (continued)**

| Offset | Register | Access | Reset | Section/page |
|---|---|---|---|---|
| 0xB28 | DDRCLKDR—DDR clock disable register | R/W | 0x0000_0000 | 23.4.1.24/23-32 |
| **Debug Control Registers** | | | | |
| 0xE00 | CLKOCR—Clock out control register | R/W | 0x0000_0000 | 23.4.1.25/23-33 |
| 0xE20 | ECMCR—ECM control register | R/W | 0x0000_0000 | 23.4.1.26/23-33 |
| 0xE60 | GCR—General control register | R/W | 0x0000_*n*000 | 23.4.1.27/23-34 |
| **SerDes1 Registers—Block Base Address 0xE_3000** | | | | |
| 0xE_3000 | SRDS1CR0—SerDes1 control register 0 | R/W | 0x1100_4430 | 23.4.1.28/2323-35 |
| 0xE_3008 | SRDS1CR2—SerDes1 control register 2 | R/W | 0x0000_0040 | 23.4.1.29/2323-37 |
| **SerDes2 Registers—Block Base Address 0xE_3100** | | | | |
| 0xE_3100 | SRDS2CR0—SerDes2 control register 0 | R/W | 0x1100_4430 | 23.4.1.30/2323-39 |
| 0xE_3104 | SRDS2CR1—SerDes2 control register 1 | R/W | 0x0000_0040 | 23.4.1.31/2323-41 |
| 0xE_3108 | SRDS2CR2—SerDes2 control register 2 | R/W | 0x0000_1C1C | 23.4.1.32/2323-42 |
| 0xE_310C | SRDS2CR3—SerDes2 control register 3 | R/W | 0x0101_0000 | 23.4.1.33/2323-44 |

## 23.4.1 Register Descriptions

This section describes the global utilities registers in detail.

### 23.4.1.1 POR PLL Status Register (PORPLLSR)

PORPLLSR, shown in Figure 23-1, contains the settings for the PLL ratios as set by the cfg_sys_pll[0:3], cfg_core_pll[0:1], and cfg_pci_clk POR configuration pins. See Section 4.4.3.1, “System PLL Ratio,” and Section 4.4.3.2, “e500 Core PLL Ratio,” for more information.

Offset 0x000 Access: Read only

| | 0 | 1 | 2 | | | | 6 | 7 | | | 10 | | | | | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | — | | DDR_Ratio | | | | | — | — | | e500_Ratio | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | *n* | *n* | *n* | *n* | *n* | 0 | 0 | 1 | *n* | *n* | *n* | *n* | *n* | *n* |

| | 16 | 17 | | | | | | | | 25 | 26 | | | | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | PCI_clk_sel | — | | | | | | | | | Plat_Ratio | | | | | 0 |
| W | | | | | | | | | | | | | | | | |
| Reset | *n* | 0 | 0 | 0 | 0 | 0 | 0 | *n* | 1 | 1 | *n* | *n* | *n* | *n* | *n* | *n* |

**Figure 23-1. POR PLL Status Register (PORPLLSR)**

Table 23-4 describes the bit settings of PORPLLSR.

**Table 23-4. PORPLLSR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–1 | — | Reserved |
| 2–6 | DDR_Ratio | Clock ratio between the DDR Complex clock and DDRCLK. Patterns not show are reserved<br>00011 3:1<br>00100 4:1<br>00110 6:1<br>01000 8:1<br>01010 10:1<br>01100 12:1<br>01110 Reserved<br>00111 Synchronous Mode-DDR Complex Clocked by CCB clock |
| 7–9 | — | Reserved |
| 10–15 | e500_Ratio | Clock ratio between the e500 core and the CCB clock. Initially, the 3 lsbs of this field correspond to the values on cfg_core_pll[0:2]. However, this register reflects the current clock ratio between the e500 core and the CCB clock, and therefore if this ratio is changed by moifying PMJCR[e500_Ratio] and executing a Deep Sleep or Jog request, then this field may not necessarily reflect the values on cfg_core_pll[0:2].<br>Patterns not shown are reserved.<br>000010 Reserved<br>000011 3:2<br>000100 2:1<br>000101 5:2<br>000110 3:1<br>000111 7:2<br>001000 4:1<br>001001 9:2 |
| 16 | PCI_clk_sel | Clock used for PCI. This bit corresponds to the values on cfg_pci_clk_sel at the negation of $\overline{\text{HRESET}}$:<br>0 PCI runs off of PCI_CLK<br>1 PCI runs off of SYSCLK |
| 17–25 | — | Reserved |
| 26–30 | Plat_Ratio | Clock ratio between the CCB (platform) clock and SYSCLK. Patterns not shown are reserved.<br>00011 3:1<br>00100 4:1<br>00101 5:1<br>00110 6:1<br>01000 8:1<br>01001 9:1<br>01010 10:1<br>01100 12:1<br>11101 20:1 |
| 31 | — | Reserved |

### 23.4.1.2 POR Boot Mode Status Register (PORBMSR)

The PORBMSR, shown in Figure 23-2, reports setting of the POR configuration pins that control the boot mode settings (described in Section 4.4.3.6, “Boot ROM Location,” Section 4.4.3.10, “CPU Boot

Configuration,” and Section 4.4.3.11, “Boot Sequencer Configuration”) and the default settings of PCI/PCI Express host/agent mode (described in Section 4.4.3.7, “Host/Agent Configuration”).

Offset 0x004 Access: Read only

| | 0 | 1 | | 3 | 4 | | | 7 | 8 | 9 | 10 | 11 | 12 | 13 | | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | BCFG | — | | | ROM_LOC | | | | — | | BSCFG | | — | HA | | |
| W | | | | | | | | | | | | | | | | |
| Reset | *n* | 0 | 0 | 0 | *n* | *n* | *n* | *n* | 0 | 0 | *n* | *n* | 0 | *n* | *n* | *n* |
| | 16 | | | | | | | | | | | | | | | 31 |
| R | — | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | *0* | *0* | *0* | 0 | 0 | *0* | *0* | *0* | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Figure 23-2. POR Boot Mode Status Register (PORBMSR)**

For more information about the PCI configurations, see Section 16.3.2.19, “PCI Bus Function Register (PBFR).” Figure 23-5 describes the bit settings of the PORBMSR.

**Table 23-5. PORBMSR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | BCFG | CPU boot configuration<br>0 The CPU is prevented from booting until configuration by an external master is complete.<br>1 The CPU is allowed to start fetching boot code. |
| 1–3 | — | Reserved |
| 4–7 | ROM_LOC | Location of boot ROM. This field reflects the values on cfg_rom_loc[0:3] at the negation of $\overline{\text{HRESET}}$.<br>0000 PCI<br>0001 PCI Express 1<br>0010 PCI Express 2<br>0011 PCI Express 3<br>0100 DDR Controller<br>0101 Reserved<br>0110 On-chip boot ROM SPI configuration<br>0111 On-chip boot ROM eSDHC configuration<br>1000 Local bus FCM: 8-bit NAND Flash small page ECC enabled<br>1001 Local bus FCM: 8-bit NAND Flash small page ECC disabled<br>1010 Local bus FCM: 8-bit NAND Flash large page ECC enabled<br>1011 Local bus FCM: 8-bit NAND Flash large page ECC disabled<br>1100 Reserved<br>1101 Local bus GPCM: 8-bit ROM<br>1110 Local bus GPCM:16-bit ROM<br>1111 Local bus GPCM: 32-bit ROM (Default) |
| 8–9 | — | Reserved |
| 10–11 | BSCFG | Boot sequencer configuration<br>00 Reserved<br>01 Boot sequencer enabled with normal $I^2C$ addressing<br>10 Boot sequencer enabled with extended $I^2C$ addressing<br>11 Boot sequencer disabled |
| 12 | — | Reserved |

**Table 23-5. PORBMSR Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 13–15 | HA | Host/agent mode configuration. When the MPC8536E is an agent on an interface, it is prevented from mastering transactions on that interface until the external host configures the interface appropriately.<br>000 Reserved<br>001 PCI Express 3 endpoint<br>010 Reserved<br>011 PCI Express 2 endpoint<br>100 Reserved<br>101 PCI Express 1 endpoint<br>110 PCI agent mode<br>111 Host mode/root complex on all interfaces |
| 16–31 | — | Reserved |

### 23.4.1.3 POR I/O Impedance Status and Control Register (PORIMPSCR)

PORIMPSCR, shown in Figure 23-3, contains the current I/O driver impedances for local bus and PCI interfaces.

Offset 0x008 Access: Mixed

| | 0–14 | 15 | 16–24 | 25–31 |
|---|---|---|---|---|
| R | — | PCI_Z | — | LBC_Z |
| W | | | | |
| Reset | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 *0* | *n* | 0 0 0 0 0 0 0 0 0 | 1 1 1 1 1 1 1 |

**Figure 23-3. POR I/O Impedance Status and Control Register (PORIMPSCR)**

The I/O impedance of local bus signals (including the local bus clock) is controlled through this register. The I/O impedance of PCI signals is controlled by POR configuration pins (described in Section 4.4.3.20, “PCI I/O Impedance”). The *MPC8536E Integrated Processor Hardware Specification* provides exact I/O impedances.

Table 23-6 describes PORIMPSCR fields.

**Table 23-6. PORIMPSCR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–14 | — | Reserved |
| 15 | PCI_Z | PCI I/O impedance<br>0 Low impedance<br>1 High impedance |
| 16–24 | — | Reserved |
| 25–31 | LBC_Z | I/O impedance for these local bus signals: LAD[0:31], LDP[0:3], LA[27:31], $\overline{\text{LCS}}$[0:7], $\overline{\text{LWE}}$[0:3], LGP[0:5], LCKE, LCLK<br>Note: Other signals use a fixed high I/O impedance<br>11111111 High impedance<br>else Low impedance |

#### 23.4.1.4 POR Device Status Register (PORDEVSR)

Shown in Figure 23-4, PORDEVSR reports other POR settings for I/O devices as described in Section 4.4.3.14, “eTSEC1 width,” Section 4.4.3.15, “eTSEC3 Width, Section 4.4.3.21, “PCI Arbiter Configuration.”

Offset 0x00C | Access: Read only

| | 0 | 1 | 2 | | 4 | 5 | 6 | 7 | 8 | 9 | 10 | | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | ECW1 | ECW2 | SRDS2_IO_SEL | | | — | ECP1 | | — | | IO_SEL | | | — | PCI_ ARB | — |
| W | | | | | | | | | | | | | | | | |
| Reset | *n* | *n* | *n* | *n* | *n* | 0 | *n* | *n* | 0 | 0 | *n* | *n* | *n* | 0 | *n* | 0 |

| | 16 | 17 | 18 | 19 | 20 | 21 | 22 | | 24 | 25 | 26 | | | | | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | PCI_S PD | SYS_ SPD | CORE_ SPD | | ECP3 | | — | | | RTYPE | — | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | *n* | *n* | n | 0 | *n* | *n* | 0 | 0 | 0 | *n* | 0 | 0 | 0 | 0 | 0 | 0 |

**Figure 23-4. POR Device Status Register (PORDEVSR)**

Table 23-7 describes the bit settings of PORDEVSR.

**Table 23-7. PORDEVSR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | ECW1 | eTSEC1 controller width (See Section 4.4.3.14, “eTSEC1 width.”)<br>0 eTSEC1 interface operates in reduced pin mode, either RTBI, RGMII, RMII<br>1 eTSEC1 interface operates in standard width TBI, GMII, MII<br>Note: eTSEC1 is always in 8-bit FIFO protocol regardingless to this field. |
| 1 | ECW2 | eTSEC3 controller width (See Section 4.4.3.15, “eTSEC3 Width.”)<br>0 eTSEC3 interface operates in reduced pin mode, either RTBI, RGMII or RMII mode<br>1 eTSEC3 interface operates in standard width MII mode<br>Note: eTSEC3 is always in 8-bit FIFO protocol regardingless to this field. |

Table 23-7. PORDEVSR Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 2–4 | SRDS2_IO_SEL | SerDes2 I/O port selection<br>000:<br>Reserved<br>001:<br>SATA1 → SerDes2 Lane A.<br>SATA2 → SerDes2 Lane B.<br>eTSEC1 and eTSEC3 Ethernet interface uses parallel interface according to POR config inputs cfg_tsec1_prtcl and cfg_tsec3_prtcl.<br>010:<br>Reserved<br>011:<br>SATA1 → SerDes2 Lane A.<br>SATA2 disabled.<br>eTSEC1 and eTSEC3 Ethernet interface uses parallel interface according to POR config inputs cfg_tsec1_prtcl and cfg_tsec3_prtcl.<br>SerDes2 Lane B disabled.<br>100:<br>SATA1 and SATA2 disabled.<br>eTSEC1 SGMII (1.25 Gbps)→ SerDes2 Lane A.<br>eTSEC3 SGMII (1.25 Gbps) → SerDes2 Lane B.<br>POR config inputs cfg_tsec1_prtcl and cfg_tsec3_prtcl should be left in their default settings.<br>101:<br>Reserved<br>110:<br>SATA1 and SATA2 disabled.<br>eTSEC1 SGMII (1.25 Gbps) → SerDes2 Lane A (POR config input cfg_tsec1_prtcl should be left in its default setting)<br>eTSEC3 parallel mode Ethernet interface (according to cfg_tsec3_prtcl).<br>SerDes2 Lane B disabled<br>111:<br>SATA1 and SATA2 disabled.<br>eTSEC1 and eTSEC3 Ethernet interface uses parallel interface according to POR config inputs cfg_tsec1_prtcl and cfg_tsec3_prtcl.<br>SerDes2 disabled |
| 5 | — | Reserved |
| 6–7 | ECP1 | eTSEC1 controller protocol (See Section 4.4.3.16, “eTSEC1 Protocol.”)<br>00 The eTSEC1 controller operates using the 8-bit FIFO protocol.<br>01 The eTSEC1 controller operates using the MII protocol (or RMII if configured in reduced mode).<br>10 The eTSEC1 controller operates using the GMII protocol (or RGMII if configured in reduced mode).<br>11 The eTSEC1 controller operates using the TBI protocol (or RTBI if configured in reduced mode). |
| 8–9 | — | Reserved |

Table 23-7. PORDEVSR Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 10–12 | IO_SEL | I/O port selection mode (See Section 4.4.3.8, "SerDes1 I/O Port Selection.")<br>• 000 Reserved<br>• 001 All 3 PCI Express ports powered down<br>• 010 PCI Express 1 (x4) (2.5 Gbps); 100 MHz reference clock<br>PCI Express 1:<br>RX lane[0:3] <- SD1_RX[0:3]<br>TX lane[0:3] -> SD1_TX[0:3]<br>• 011 PCI Express 1 (x8) (2.5 Gbps); 100 MHZ reference clock<br>PCI Express 1:<br>RX lane[0:7] <- SD1_RX[0:7]<br>TX lane[0:7] -> SD1_TX[0:7]<br>• 100 Reserved<br>• 101 PCI Express 1 (x4) (2.5 Gbps), PCI Express 2 (x4) (2.5 Gbps); 100 MHz reference clock<br>PCI Express 1:<br>RX lane[0:3] <- SD1_RX[0:3]<br>TX lane[0:3] -> SD1_TX[0:3]<br>PCI Express 2:<br>RX lane[0:3] <- SD1_RX[4:7]<br>TX lane[0:3] -> SD1_TX[4:7]<br>• 110 Reserved<br>• 111 PCI Express 1 (x4) (2.5 Gbps), PCI Express 2 (2x) (2.5 Gbps), PCI Express 3 (2x) (2.5 Gbps); 100 MHz reference clock<br>PCI Express 1:<br>RX lane[0:3] <- SD1_RX[0:3]<br>TX lane[0:3] -> SD1_TX[0:3]<br>PCI Express 2:<br>RX lane[0:1] <- SD1_RX[4:5]<br>TX lane[0:1] -> SD1_TX[4:5]<br>PCI Express 3:<br>RX lane[0:1] <- SD1_RX[6:7]<br>TX lane[0:1] -> SD1_TX[6:7] |
| 13 | — | Reserved |
| 14 | PCI_ARB | PCI arbiter enable (See Section 4.4.3.21, "PCI Arbiter Configuration.")<br>0 PCI arbiter is disabled<br>1 PCI arbiter is enabled |
| 15 | — | Reserved |
| 16 | PCI_SPD | PCI clock speed (See Section 4.4.3.19, "PCI Speed Configuration.")<br>0 PCI set for low speed operation—PCI below 33MHz<br>1 PCI set for normal speed operation—PCI at or above 33MHz |
| 17 | SYS_SPD | System clock speed (See Section 4.4.3.19, "PCI Speed Configuration")<br>0 SYSCLK frequency at or below 66MHz<br>1 SYSCLK frequency above 66MHz |

**Table 23-7. PORDEVSR Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 18 | CORE_SPD | Core clock speed (See Section 4.4.3.5, “Core Speed Configuration”)<br>This field reflects the current core speed, and therefore if this is changed by modifying PMJCR[CORE_SPD] and executing a Deep Sleep or Jog request, then CORE_SPD may not necessarily reflect the values at POR.<br>0 Core frequency at or below 800 MHz<br>1 Core frequency above 800 MHz |
| 19 | — | Reserved |
| 20–21 | ECP3 | eTSEC3 controller protocol (See Section 4.4.3.17, “eTSEC3 Protocol.”)<br>00 The eTSEC3 controller operates using the 8-bit FIFO protocol.<br>01 The eTSEC3 controller operates using the MII protocol (or RMII if configured in reduced mode).<br>10 The eTSEC3 controller operates using the RGMII protocol.<br>11 The eTSEC3 controller operates using the RTBI protocol . |
| 22–24 | — | Reserved |
| 25 | RTYPE | DRAM Type for DDR Controllers (See Section 4.4.3.12, “DDR SDRAM Type.”)<br>0 DDR3 (1.5 V, CKE low at reset)<br>1 DDR2 (1.8 V, CKE low at reset) |
| 26–31 | — | Reserved |

### 23.4.1.5 POR Debug Mode Status Register (PORDBGMSR)

PORDBGMSR, shown in Figure 23-5, holds debug mode settings from the POR configuration pins as described in Section 4.4.3.22, “Memory Debug Configuration,” and Section 4.4.3.23, “DDR Debug Configuration.”

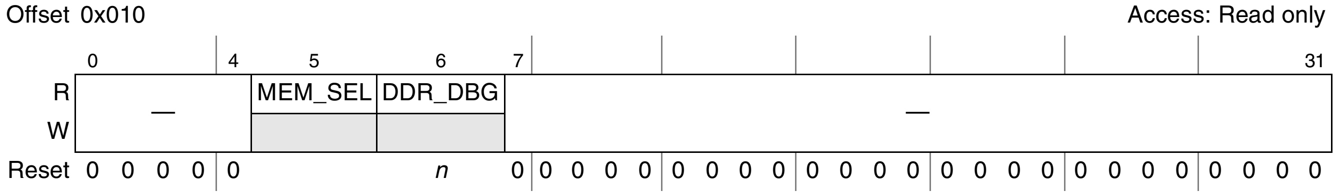


**Figure 23-5. POR Debug Mode Status Register (PORDBGMSR)**

Table 23-8 describes the bit settings of PORDBGMSR.

**Table 23-8. PORDBGMSR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–4 | — | Reserved |
| 5 | MEM_SEL | Memory select. Indicates which controller is driving MSRCID[0:4] and MDVAL.<br>0 Local bus controller is driving debug information<br>1 DDR SDRAM controller is driving debug information |
| 6 | DDR_DBG | DDR debug configuration<br>0 SourceID and data valid information is being driven on ECC pins of DDR SDRAM interface<br>1 Normal mode. ECC information is being driven on ECC pins of DDR SDRAM interface |
| 7–31 | — | Reserved |

### 23.4.1.6 POR Device Status Register 2 (PORDEVSR2)

Shown in Figure 23-6,the PORDEVSR2 reports POR settings as described in Section 4.4.3.13, “Serdes 2 Reference Clock Configuration.”

Offset 0x014 Access: Read only

| | 0–17 | 18–19 | 20–31 |
|---|---|---|---|
| R | — | SRDS2_REFCLK | — |
| W | | | |
| Reset | *n* 0 0 0 *n n n n* 0 0 0 0 0 0 0 0 0 0 | 0 0 | 0 0 0 0 0 0 0 1 1 1 1 1 |

**Figure 23-6. POR Device Status Register 2 (PORDEVSR2)**

Table 23-9 describes the bit settings of PORDEVSR2.

**Table 23-9. PORDEVSR2 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–17 | — | Reserved |
| 18–19 | SRDS2_REFCLK | SerDes2 reference clock<br>00 Reserved<br>01 When configured for SATA: SerDes2 expects a 150 MHz reference clock frequency. This should not be used when SerDes2 is configured for SGMII.<br>10 SerDes2 expects a 125 MHz reference clock frequency for either SATA or SGMII functionality.<br>11 SerDes2 expects a 100 MHz reference clock frequency for either SATA or SGMII functionality. (default). |
| 21–31 | — | Reserved |

### 23.4.1.7 General-Purpose POR Configuration Register (GPPORCR)

GPPORCR stores the value sampled from the local bus address/data signals, LAD[0:31], during POR, as described in Section 4.4.3.24, “General-Purpose POR Configuration.” Software can use this value to inform the operating system about initial system configuration. Typical interpretations include circuit board type, board ID number, or a list of available peripherals.

GPPORCR is shown in Figure 23-7.

Offset 0x020 Access: Read Only

| | 0–15 | 16–31 |
|---|---|---|
| R | POR_CFG_VEC | POR_CFG_VEC |
| W | | |
| Reset | *n n n n n n n n n n n n n n n n* | *n n n n n n n n n n n n n n n n* |

**Figure 23-7. POR Configuration Register (GPPORCR)**

Table 23-10 describes the bit settings of GPPORCR.

**Table 23-10. GPPORCR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–31 | POR_CFG_VEC | General-purpose POR configuration vector sampled from local bus address/data signals at the negation of $\overline{\text{HRESET}}$. Note that if nothing is driven on these signals during reset, the value of this register is indeterminate. |

### 23.4.1.8 General Configuration Register (GENCFGR)

GENCFGR is shown in Figure 23-8.

Offset 0x030 Access: Read Only

| | 0 1 | 2 | 3 31 |
|---|---|---|---|
| R<br>W | — | SDHC_WP_INV | — |
| Reset | All zeros | | |

**Figure 23-8. General Configuration Register (GENCFGR)**

Table 23-11 describes the bit settings of GENCFGR.

**Table 23-11. GENCFGR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–1 | — | Reserved |
| 2 | SDHC_WP_INV | Secure digital WP polarity<br>0 SDHC_WP is active high (0=write enabled, 1=write protected).<br>1 SDHC_WP is active low (0=write protected, 1=write enabled).<br>Note: This bit should be set to 1 if PMUXCR[SDHC_WP]=0 (SDHC_WP not exposed to pins) and eSDHC write functionality is required. |
| 3–31 | — | Reserved |

### 23.4.1.9 Alternate Function Signal Multiplex Control Register (PMUXCR)

Shown in Figure 23-9, PMUXCR contains bits that enable DMA channels 0, 1, 2 and 3 which exist as alternate functions on GPIO, on local bus chip select pins $\overline{\text{LCS}}$[5:7], and interrupt input pins IRQ[9:11], respectively. Specifically, DMA request, acknowledge, and done signals comprise the secondary functions for the associated IRQ and local bus chip select signals.

It contains also the SPI/ eSDHC, GPIO/eSDHC, GPIO/USB1, GPIO/USB2 and GPIO/PCI controls for pinmux.

Offset 0x060 Access: Read/Write

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7–13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | SD_DATA | SDHC_CD | SDHC_WP | PCI_REQGNT3 | PCI_REQGNT4 | USB1 | USB2 | — | DMA0 | DMA2 |
| Reset | All zeros | | | | | | | | | |

| | 16–29 | 30 | 31 |
|---|---|---|---|
| R<br>W | — | DMA1 | DMA3 |
| Reset | All zeros | | |

**Figure 23-9. Alternate Function Pin Multiplex Control Register (PMUXCR)**

Table 23-12 describes the bit settings of PMUXCR.

**Table 23-12. PMUXCR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | SD_DATA | Enables SD_DATA[4:7] signals<br>0 SD_DATA[4:7] is not exposed to pins; the pins retain the primary function as SPI.<br>1 SD_DATA[4:7] is exposed to pins as follows:<br>$\overline{\text{SPI\_CS}}$[0] functions as SDHC_DAT[4]<br>$\overline{\text{SPI\_CS}}$[1] functions as SDHC_DAT[5]<br>$\overline{\text{SPI\_CS}}$[2] functions as SDHC_DAT[6]<br>$\overline{\text{SPI\_CS}}$[3] functions as SDHC_DAT[7] |
| 1 | SDHC_CD | Enables $\overline{\text{SDHC\_CD}}$ signals<br>0 $\overline{\text{SDHC\_CD}}$ is not exposed to pins; the pins retain the primary function as GPIO.<br>1 $\overline{\text{SDHC\_CD}}$ is exposed to pins as follows:<br>GPIO[4] functions as $\overline{\text{SDHC\_CD}}$ |
| 2 | SDHC_WP | Enables SDHC_WP signals<br>0 SDHC_WP is not exposed to pins; the pins retain the primary function as GPIO.<br>1 SDHC_WP is exposed to pins as follows:<br>GPIO[5] functions as SDHC_WP<br>Note: If PMUXCR[SDHC_WP]=0 and eSDHC write functionality is required, then GENCFGR[SDHC_WP_INV] should be set to 1. |
| 3 | PCI_REQGNT3 | Enables $\overline{\text{PCI\_REQ}}$[3] and $\overline{\text{PCI\_GNT}}$[3] signals<br>0 $\overline{\text{PCI\_REQ}}$[3] and $\overline{\text{PCI\_GNT}}$[3] are not exposed to pins; the pins retain the primary function as GPIO.<br>1 $\overline{\text{PCI\_REQ}}$[3] and $\overline{\text{PCI\_GNT}}$[3] are exposed to pins as follows:<br>GPIO[0] functions as $\overline{\text{PCI\_REQ}}$[3]<br>GPIO[2] functions as $\overline{\text{PCI\_GNT}}$[3] |
| 4 | PCI_REQGNT4 | Enables $\overline{\text{PCI\_REQ}}$[4] and $\overline{\text{PCI\_GNT}}$[4] signals<br>0 $\overline{\text{PCI\_REQ}}$[4] and $\overline{\text{PCI\_GNT}}$[4] are not exposed to pins; the pins retain the primary function as GPIO.<br>1 $\overline{\text{PCI\_REQ}}$[4] and $\overline{\text{PCI\_GNT}}$[4] are exposed to pins as follows:<br>GPIO[1] functions as $\overline{\text{PCI\_REQ}}$[4]<br>GPIO[3] functions as $\overline{\text{PCI\_GNT}}$[4] |
| 5 | USB1 | Enables USB1_PCTL0 and USB1_PCTL1 signals<br>0 USB1_PCTL0 and USB1_PCTL1 are not exposed to pins; the pins retain the primary function as GPIO.<br>1 USB1_PCTL0 and USB1_PCTL1 are exposed to pins as follows:<br>GPIO[6] functions as USB1_PCTL0<br>GPIO[7] functions as USB1_PCTL1 |

Table 23-12. PMUXCR Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 6 | USB2 | Enables USB2_PCTL0 and USB2_PCTL1 signals<br>0 USB2_PCTL0 and USB2_PCTL1 are not exposed to pins; the pins retain the primary function as GPIO.<br>1 USB2_PCTL0 and USB2_PCTL1 are exposed to pins as follows:<br>GPIO[8] functions as USB2_PCTL0<br>GPIO[9] functions as USB2_PCTL1 |
| 7–13 | — | Reserved |
| 14 | DMA0 | Enables DMA channel 0 signals.<br>0 DMA channel 0 is not exposed to pins; the pins retain their primary function as GPIO.<br>1 DMA channel 0 is exposed to pins as follows:<br>GPIO[10] functions as DMA_DREQ0<br>GPIO[12] functions as DMA_DACK0<br>GPIO[14] funnctions as DMA_DDONE0 |
| 15 | DMA2 | Enables DMA channel 2 signals.<br>0 DMA channel 2 is not exposed to pins; the pins retain their primary function as local bus chip selects<br>1 DMA channel 2 is exposed to pins as follows:<br>LCS5 functions as DMA_DREQ2<br>LCS6 functions as DMA_DACK2<br>LCS7 functions as DMA_DDONE2 |
| 16–29 | — | Reserved |
| 30 | DMA1 | Enables DMA channel 1 signals.<br>0 DMA channel 1 is not exposed to pins; the pins retain their primary function as GPIO<br>1 DMA channel 1 is exposed to pins as follows:<br>GPIO[11] functions as DMA_DREQ1<br>GPIO[13] functions as DMA_DACK1<br>GPIO[15] funnctions as DMA_DDONE1 |
| 31 | DMA3 | Enables DMA channel 3 signals.<br>0 DMA channel 3 is not exposed to pins; the pins retain their primary function as interrupt requests.<br>1 DMA channel 3 is exposed to pins as follows:<br>IRQ9 functions as DMA_DREQ3<br>IRQ10 functions as DMA_DACK3<br>IRQ11 functions as DMA_DDONE3 |

### 23.4.1.10 Device Disable Register (DEVDISR)

DEVDISR, shown in Figure 23-10, contains disable bits for various MPC8536E functional blocks.

Offset 0x070 Access: Read/Write

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | PCI | — | PCIE1 | — | ELBC | PCIE2 | PCIE3 | SEC | USB1 | USB2 | USB3 | L2 | eSDHC | SATA1 | — | SPI |
| Reset | *n* | 0 | *n* | 0 | 0 | *n* | *n* | 0 | 0 | 0 | 0 | 0 | 0 | *n* | 0 | 0 |

| | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | DDR | E500 | TB | — | SATA2 | DMA | — | SRDS2 | eTSEC1 | — | eTSEC3 | — | | I2C | DUART | SRDS1 |
| Reset | 0 | 0 | 0 | 0 | *n* | 0 | 0 | *n* | 0 | 0 | 0 | 0 | 0 | 0 | 0 | *n* |

**Figure 23-10. Device Disable Register (DEVDISR)**

Note that bits with a reset value of *n* depend on the state of POR configuration signals at reset.

All functional blocks are enabled after reset; unneeded blocks can be disabled to reduce power consumption. Blocks disabled by DEVDISR must not be re-enabled without a hard reset. Section 23.5.1.5, "Shutting Down Unused Blocks," has more information on the use of DEVDISR.

Table 23-13 describes DEVDISR fields.

**Table 23-13. DEVDISR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | PCI | PCI controller disable.<br>0 PCI controller enable<br>1 PCI controller disable |
| 1 | — | Reserved. Should be cleared. |
| 2 | PCIE1 | PCI Express 1 controller disable.<br>0 PCI Express 1 controller enable<br>1 PCI Express 1 controller disable |
| 3 | — | Reserved. Should be cleared. |
| 4 | ELBC | Enhanced local bus controller disable.<br>0 Enhanced local bus controller enable<br>1 Enhanced local bus controller disable |
| 5 | PCIE2 | PCI Express 2 controller disable.<br>0 PCI Express 2 controller enable<br>1 PCI Express 2 controller disable |
| 6 | PCIE3 | PCI Express 3 controller disable.<br>0 PCI Express 3 controller enable<br>1 PCI Express 3 controller disable |
| 7 | SEC | Security disable controller.<br>0 Security enable<br>1 Security disable |
| 8 | USB1 | USB 1 controller disable.<br>0 USB 1 enable<br>1 USB 1 disable |

**Table 23-13. DEVDISR Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 9 | USB2 | USB 2 controller disable.<br>0 USB 2 enable<br>1 USB 2 disable |
| 10 | USB3 | USB 3 controller disable.<br>0 USB 3 enable<br>1 USB 3 disable |
| 11 | L2 | L2 controller disable.<br>0 L2 enable<br>1 L2 disable |
| 12 | eSDHC | eSDHC controller disable.<br>0 eSDHC enable<br>1 eSDHC disable |
| 13 | SATA1 | SATA 1 controller disable.<br>0 SATA 1 enable<br>1 SATA 1 disable |
| 14 | — | Reserved |
| 15 | SPI | SPI controller disable.<br>0 SPI enable<br>1 SPI disable |
| 16 | DDR | DDR SDRAM controller disable.<br>0 DDR controller enable<br>1 DDR controller disable |
| 17 | E500 | e500 core disable.<br>0 e500 core enable<br>1 e500 core disable. Places the core in the core_stopped state in which it does not respond to interrupts. Equivalent to nap mode. Instruction fetching is stopped, snooping is disabled, and clocks are shut down to all functional units of the core including the timer facilities. |
| 18 | TB | Time base (timer facilities) of the e500 core disable.<br>0 Timer facilities enabled<br>1 Timer facilities disabled. |
| 19 | — | Reserved |
| 20 | SATA2 | SATA 2 controller disable.<br>0 SATA 2 enable<br>1 SATA 2 disable |
| 21 | DMA | DMA controller disabled.<br>0 DMA controller enabled<br>1 DMA controller disabled |
| 22 | — | Reserved |
| 23 | SRDS2 | SerDes 2 disabled.<br>0 SerDes 2 enabled<br>1 SerDes 2 disabled |

Table 23-13. DEVDISR Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 24 | eTSEC1 | Three-speed Ethernet controller 1 disable.<br>0 eTSEC 1 enabled<br>1 eTSEC 1 disabled |
| 25 | — | Reserved |
| 26 | eTSEC3 | Three-speed Ethernet controller 3 disable.<br>0 eTSEC 3 enabled<br>1 eTSEC 3 disabled |
| 27–28 | — | Reserved |
| 29 | I2C | I2C controllers disabled.<br>0 I2C controllers enabled<br>1 I2C controllers disabled |
| 30 | DUART | Dual UART controller disabled.<br>0 DUART enabled<br>1 DUART disabled |
| 31 | SRDS1 | SerDes 1 disabled.<br>0 SerDes 1 enabled<br>1 SerDes 1 disabled |

### 23.4.1.11 Power Management Jog Control Register (PMJCR)

The power management jog control register (PMJCR) is shown in Figure 23-12.

Offset 0x07C

Access: Read/Write

| | 0 | | | | | | | | | 9 | 10 | | | | | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | | | | | | | | | | e500_Ratio | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | *n* | *n* | *n* | *n* | *n* | *n* |

| | 16 | 17 | 18 | 19 | | | | | | | | | | | | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | | CORE_SPD | | — | | | | | | | | | | | |
| Reset | All zeros | | | | | | | | | | | | | | | |

Figure 23-11. Power Management Jog Control Register (PMJCR)

Table 23-14 describes PMJCR fields.

Table 23-14. PMJCR Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0-9 | — | Reserved |
| 10-15 | e500_Ratio | Requested clock ratio between e500 core and CCB clock:<br>000010 Reserved<br>000011 3:2<br>000100 2:1<br>000101 5:2<br>000110 3:1<br>000111 7:2<br>001000 4:1<br>001001 Reserved<br>Default value is the same as PORPLLSR[e500_Ratio].<br>The value written to this register takes effect when the system wakes from deep sleep or when a jog mode request is initiated.<br>Note that if this register has been written by software, but deep sleep has not occurred , then the value in this register may not necessarily reflect the current clock ratio between the e500 core and CCB clock. |
| 16-17 | — | Reserved |
| 18 | CORE_SPD | Requested core clock speed<br>0 Core frequency at or below 800 MHz<br>1 Core frequency above 800 MHz<br>Default value is the same as PORDEVSR[CORE_SPD].<br>The value written to this register takes effect when the system wakes from deep sleep or when a jog mode request is initiated.<br>Note that if this register has been written by software, but deep sleep has not occurred , then the value in this register may not necessarily reflect the current core clock speed. However, this value must be consistent at all times with the value programmed in PMJCR[e500_Ratio]. |
| 19-31 | — | Reserved |

### 23.4.1.12 Power Management Control and Status Register (POWMGTCSR)

The power management control and status register (POWMGTCSR) is shown in Figure 23-12, contains bits for placing the MPC8536E into low power states and for controlling when it wakes up. It also contains power management status bits.

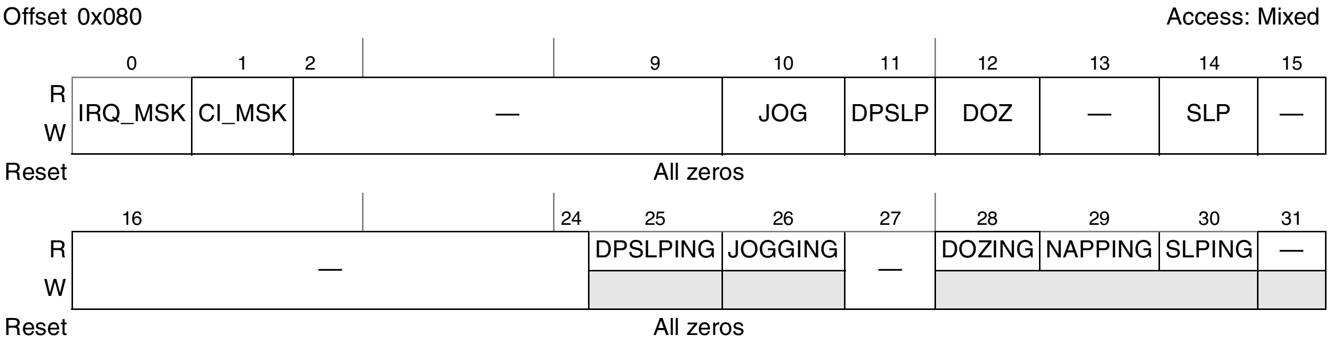


**Figure 23-12. Power Management Control and Status Register (POWMGTCSR)**

Table 23-15 describes the bit settings of POWMGTCSR.

**Table 23-15. POWMGTCSR Field Descriptions**

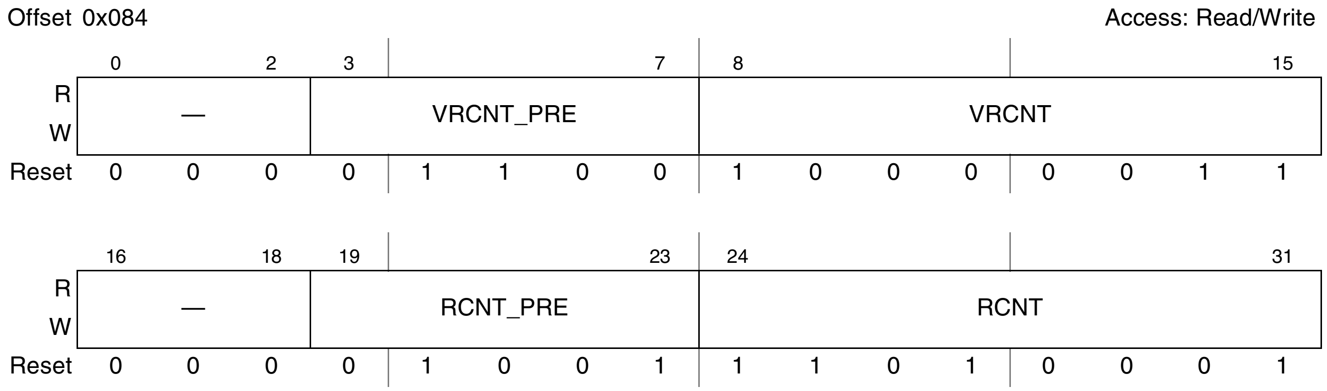
| Bits | Name | Description |
|---|---|---|
| 0 | IRQ_MSK | Interrupt input mask (e500)<br>0 Interrupts cause the device to wake up from a low-power state.<br>1 Interrupts are masked as a wake-up condition. The device remains in a low-power state despite the presence of an interrupt request. |
| 1 | CI_MSK | Critical interrupt input mask (e500)<br>0 Critical interrupts cause the device to wake up from a low power state.<br>1 Critical interrupts are masked as a wake-up condition. The device remains in a low-power state despite the presence of a critical interrupt. |
| 2–9 | — | Reserved |
| 10 | JOG | Jog mode<br>0 No Jog request<br>1 Jog request. (The user should not issue a Jog request when in or when entering Doze/Snap/Sleep/DeepSleep) |
| 11 | DPSLP | Deep sleep mode<br>0 No request to remove power to the core and the L2 in low power mode.<br>1 Remove power to the core and the L2 in low power mode (deep sleep mode). |
| 12 | DOZ | Doze mode<br>0 No request to put device in doze mode. Note that this bit is automatically cleared on MCP, UDE, SRESET, core_tbint (from the core) and also int and cint if not masked.<br>1 Device is to be placed in doze mode. Instruction fetching is halted in the e500 core. Note that this bit is logically ORed with HID0[DOZE]. |
| 13 | — | Reserved |
| 14 | SLP | Sleep mode<br>0 No request to put device in sleep mode.<br>1 Device is to be placed in sleep mode. Instruction fetching is halted, snooping of L1 caches is disabled, and most functional blocks are shut down in both the e500 core and the system logic. |

Table 23-15. POWMGTCSR Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 15–24 | — | Reserved |
| 25 | DPSLPING | Deep sleep status<br>0 Device is not attempting to reach deep sleep mode.<br>1 The device is attempting to DEEP SLEEP because POWMGTCSR[DPSLP] is set. Most functional blocks in the core and device are shut down or are attempting to shut down. |
| 26 | JOGGING | Jog status<br>0 No Jog request<br>1 Device is in jog mode. Functional blocks in the core and device are shut down or are attempting to shut down. |
| 27 | — | Reserved |
| 28 | DOZING | Doze status<br>0 Device is not in doze mode.<br>1 The MPC8536E is in doze mode because POWMGTCSR[DOZ] is set or because HID0[DOZE] and MSR[WE] (in the e500 core) are set. The core has halted instruction fetching, but all other functional blocks in the core and device are running. |
| 29 | NAPPING | Nap status<br>0 Device is not in nap mode.<br>1 The MPC8536E is in nap mode because HID0[NAP] and MSR[WE] are set. The core has halted instruction fetching, snooping of the L1 caches is disabled, and all of the core functional units except the timer facilities are shut down. All functional blocks in the device are running. |
| 30 | SLPING | Sleep status<br>0 Device is not attempting to reach sleep mode.<br>1 The device is attempting to SLEEP because POWMGTCSR[SLP] is set or because HID0[SLEEP] and MSR[WE] (in the e500 core) are set. Most functional blocks in the core and device are shut down or are attempting to shut down. |
| 31 | — | Reserved |

### 23.4.1.13 Power Management Reset Counters Configuration Register (PMRCCR)

The power management reset counter configuration register (PMRCCR), shown in Figure 23-13, contains bits that configure the reset counter used in deep sleep mode.



Offset 0x084

Access: Read/Write

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | | | VRCNT_PRE | | | | | VRCNT | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

| | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | | | RCNT_PRE | | | | | RCNT | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |

Figure 23-13. Power Management Reset Counters Configuration Register (PMRCCR)

Table 23-16 describes PMRCCR fields.

**Table 23-16. PMRCCR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–2 | — | Reserved |
| 3–7 | VRCNT_PRE | Voltage ramp-up count prescaler.<br>This field specifies the prescaler for the voltage reset counter. Prescale value is $2^{VRCNT\_PRE–1}$.<br>0x00 Reserved<br>0x01 1<br>0x02 2<br>0x03 4<br>0x04 8<br>...<br>0x0C 2048 (default)<br>...<br>0x1F 1,073,741,824 |
| 8–15 | VRCNT | Voltage ramp-up count value.<br>When waking up from deep sleep, power (VDD) is re-applied to a portion of the die. This value determines the duration for VDD to become stable after POWER_OK is asserted before enabling the e500 core PLL.<br>In systems where POWER_OK is not provided externally and is tied active, this must represent the full voltage ramp-up time.<br>In systems where POWER_OK is provided externally, this represents an optional additional delay to wait after assertion of POWER_OK. This can be used as extra margin to guarantee voltage stabilisation.<br>When the VRCNT counter reaches 0, the core PLL is enabled and the RCNT counter begins to decrement to initiate the core and L2 reset sequence. If the core PLL is enabled before the power is stable it might become unpredictable and might not lock.<br>Software needs to set the VRCNT value based on the platform clock frequency and the amount of time required for the VDD power supply to ramp.<br>The default values for VRCNT_PRE and VRCNT are set to provide a minimum of 500 µs voltage ramp-up time (when used with platform clock up to 533 MHz). |
| 16–18 | — | Reserved |

Table 23-16. PMRCCR Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 19–23 | RCNT_PRE | Reset count prescaler.<br>This field specifies the prescaler for the reset counter. Prescale value is $2^{\text{RCNT\_PRE}-1}$.<br>0x00 Reserved<br>0x01 1<br>0x02 2<br>0x03 4<br>0x04 8<br>...<br>0x09 256 (default)<br>...<br>0x1F 1,073,741,824 |
| 24–31 | RCNT | Reset count value.<br>When waking up from deep sleep power (VDD) is re-applied to a portion of the die. This value determines the duration of the reset signal applied to this logic when power is re-applied. If POWMGTCSR[DPSLP] = 0, this field has no effect. Reset is applied to the powered-off region upon entering deep sleep. The RCNT value is copied into a decrementer that counts down at the rate specified by the pre-scaler RCNT_FDR. When a wakeup event occurs, PMC will wait for the POWER_OK signal to be asserted and the VRCNT counter to expire then begin decrementing the reset counter. When the counter reaches 0 reset is removed. See also the description of GCR[DEEPSLEEP_Z] in Section 23.4.1.27, "General Control Register (GCR)."<br>WARNING: If the values placed in this register is too small, the reset may not assert long enough to allow the chip to function properly. The default value is larger than the time it takes for the e500 PLLs to re-lock.<br>The default values for RCNT_PRE and RCNT are set to provide a minimum of 100 µs reset time (when used with platform clock up to 533 MHz). |

### 23.4.1.14 Power Management Power Down Counters Configuration Register (PMPDCCR)

The power management power down counter configuration register (PMPDCCR) is shown in Figure 23-14. The register contains bits that configure the power down counter used in deep sleep mode.

Offset 0x088 Access: Read/Write

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | | | PDCNT_PRE | | | | | PDCNT | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |

| | 16–31 |
|---|---|
| R<br>W | — |
| Reset | All zeros |

Figure 23-14. Power Management Power Down Counter Configuration Register (PMPDCCR)

Table 23-17 describes PMPDCCR fields.

**Table 23-17. PMPDCCR Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–2 | — | Reserved |
| 3–7 | PDCNT_PRE | Power down count prescaler.<br>This field specifies the prescaler for the power down counter. Prescale value is $2^{\text{PDCNT\_PRE}-1}$.<br>0x00 Reserved<br>0x01 1<br>0x02 2<br>0x03 4<br>0x04 8<br>...<br>0x08 128 (default)<br>...<br>0x1F 1,073,741,824 |
| 8–15 | PDCNT | Power down count value.<br>This counter establishes a minimum time for which power can be removed to the VDD supply during deep sleep. When the MPC8356E enters deep sleep the POWER_EN signal toggles low. At this point this counter is loaded with the PDCNT value and begins to decrement at the rate specified by the pre-scaler PDCNT_FDR. PMC will not respond to a wakeup request and toggle POWER_EN high until this counter has expired. The count value is reloaded each time the VDD power is removed. If POWMGCR1[DPSLP] = 0 this field has no effect. Software needs to set this register based on the PMC clock frequency and the requirements of the power supply.<br>WARNING: If the value placed in this register is too small, the power supply may cycle too quickly and the chip may not function properly.<br>The default values for PDCNT_PRE and PDCNT are set to provide a minimum of 50 µs voltage ramp-down time (when used with platform clock rates up to 533 MHz). |
| 16–31 | — | Reserved |

### 23.4.1.15 Power Management Clock Disable Register (PMCDR)

The power management clock disable register (PMCDR), shown in Figure 23-15, contains bits to disable various MPC8536E functional blocks. The register determines the blocks which will shut down the clock in sleep/deep sleep power states.

Offset 0x08C Access: Read/Write

|  | 0 |  |  |  |  |  |  | 7 | 8 | 9 | 10 | 11 |  |  |  | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — |  |  |  |  |  |  |  | USB1 | USB2 | USB3 | — |  |  |  |  |
| Reset | All zeros |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

|  | 16 |  |  | 19 | 20 | 21 |  | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — |  |  |  | SAP | — |  |  | eTSEC1 | — | eTSEC3 | — |  |  |  |  |
| Reset | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Figure 23-15. Power Management Clock Disable Register (PMCDR)**

Table 23-18 describes PMCDR fields.

Table 23-18. PMCDR Register Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–7 | — | Reserved |
| 8 | USB1 | USB 1 controller disable clock in low power modes.<br>0 USB enable clock<br>1 USB disable clock |
| 9 | USB2 | USB 2 controller disable clock in low power modes.<br>0 USB enable clock<br>1 USB disable clock |
| 10 | USB3 | USB 3 controller disable clock in low power modes.<br>0 USB enable clock<br>1 USB disable clock |
| 11–19 | — | Reserved |
| 20 | SAP | Debug mode.<br>0 Enable memory DRAM and Local Bus to be accessed by SAP during sleep or deep sleep<br>1 Disable memory DRAM, Local Bus and SAP |
| 21–23 | — | Reserved |
| 24 | TSEC1 | Three-speed Ethernet controller 1 disable clock in low power modes.<br>0 eTSEC1 enabled clock<br>1 eTSEC1 disabled clock<br>This bit is used in conjunction with tsec1_mac_mpen input signal to the PMC to determine whether wake on Magic packet or wake on ARP packet is selected.<br>If wake on ARP packet is selected, the clocks to eTSEC1, AXI2CU, ECM, DDRTQ and DDR controller will stay ON in low power modes. |
| 25 | — | Reserved |
| 26 | TSEC3 | Three-speed Ethernet controller 3 disable clock in low power modes.<br>0 eTSEC3 enabled clock<br>1 eTSEC3 disabled clock<br>This bit is used in conjunction with tsec3_mac_mpen input signal to the PMC to determine whether wake on Magic packet or wake on ARP packet is selected.<br>If wake on ARP packet is selected, the clocks to eTSEC3, AXI2CU, ECM, DDRTQ and DDR controller will stay ON in low power modes. |
| 27–31 | — | Reserved |

### 23.4.1.16 Machine Check Summary Register (MCPSUMR)

Shown in Figure 23-16, MCPSUMR contains bits summarizing some of the sources of a pending machine check interrupt. All MCPSUMR bits function as write-1-to-clear.

NOTE

> Register fields designated as write-1-to-clear are cleared only by writing ones to them. Writing zeros to them has no effect.

Note that other conditions can cause a machine check condition not summarized in MCPSUMR. For example, uncorrectable read errors cause the assertion of *core_fault_in*, which may directly cause a machine check (if HID1[RFXE] = 1). If RFXE = 0, the assertion of *core_fault_in* does not directly cause a machine check interrupt, but must be handled by the block that generated the error. For more information about RFXE, see Section 5.3, “Summary of Core Integratation Details,” and the section on HID1 in the register model chapter of the *PowerPC e500 Core Family Reference Manual*.

Offset 0x090 Access: w1c

| | 0–28 | 29 | 30 | 31 |
|---|---|---|---|---|
| R | — | WRS | SRESET | MCP_IN |
| W | | w1c | w1c | w1c |
| Reset | All zeros | | | |

**Figure 23-16. Machine Check Summary Register (MCPSUMR)**

Table 23-19 describes the bit settings of MCPSUMR.

**Table 23-19. MCPSUMR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–28 | — | Reserved |
| 29 | WRS | Watchdog timer machine check<br>0 Machine check exception was not caused by watchdog timer.<br>1 Machine check was caused by a soft reset condition from the e500 watchdog timer as configured in the core’s TSR. Specifically, TSR[WRS] = 01 and a watchdog reset condition occurred. |
| 30 | SRESET | Soft reset machine check<br>0 Machine check exception was not caused by $\overline{\text{SRESET}}$ assertion.<br>1 Machine check exception was caused by the assertion of the $\overline{\text{SRESET}}$ input signal. |
| 31 | MCP_IN | $\overline{\text{MCP}}$ signal asserted<br>0 Machine check exception was not caused by $\overline{\text{MCP}}$ assertion.<br>1 Machine check exception was caused by the assertion of the $\overline{\text{MCP}}$ input signal. |

## 23.4.1.17 Reset Request Status and Control Register (RSTRSCR)

Shown in Figure 23-17, RSTRSCR contains the status for boot sequencer, watchdog timer, and a software settable reset request bit

Offset 0x094 Access: Read Only

| | 0–8 | 9 | 10 | 11 | 12 | 13–31 |
|---|---|---|---|---|---|---|
| R | | NFLSH_RR | BS_RR | WDT0_RR | SW_RR | |
| W | | | | | | |
| Reset | All zeros | | | | | |

**Figure 23-17. Reset Request Status and Control Register (RSTRSCR)**

Table 23-20 describes the bit settings of RSTRSCR.

Table 23-20. RSTRSCR Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–8 | — | Reserved |
| 9 | NFLSH_RR | NAND Flash ECC error during boot reset request. |
| 10 | BS_RR | Boot sequence reset request |
| 11 | WDT_RR | Watchdog timer reset request in the core . Occurs when TSR[WRS] = 10 for the core and a watchdog reset condition is reached. |
| 12 | SW_RR | Software settable reset request |
| 13–31 | — | Reserved |

### 23.4.1.18 Exception Reset Control Register (ECTRSTCR)

Shown in Figure 23-18, the ECTRSTCR contains control bits for the exception reset of core in response to checkstop.

Offset 0x098 Access: Mixed

| | 0 | 1 3 | 4 | 5 31 |
|---|---|---|---|---|
| R<br>W | RST_CKSTP_EN | — | CKSTP_OUT_DIS | — |
| Reset | All zeros | | | |

Figure 23-18. Exception Reset Control Register (ECTRSTCR)

Table 23-21 describes the bit settings of ECTRSTCR.

Table 23-21. ECTRSTCR Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0 | RST_CKSTP_EN | Enable automatic reset of core in response to core checkstop |
| 1–3 | — | Reserved |
| 4 | CKSTP_OUT_DIS | Disable assertion of $\overline{\text{CKSTP\_OUT}}$ pin |
| 5–31 | — | Reserved |

### 23.4.1.19 Automatic Reset Status Register (AUTORSTSR)

Shown in Figure 23-19, the AUTORSTSR contains the automatic reset status bits for core 0 and core 1.

Offset 0x09C Access: Mixed

| | 0 | 1 3 | 4 | 5 7 | 8 | 9 11 | 12 | 13 15 | 16 | 17 19 | 20 | 21 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | RST_ CKSTP | — | RST_ WRS | — | RST_ MPIC | — | RST_ CORE | — | READY | — | RST_ DPSLP | — |
| W | w1c | | w1c | | w1c | | w1c | | | | w1c | |
| Reset | All zeros | | | | | | | | | | | |

Figure 23-19. Automatic Reset Status Register (AUTORSTSR)

Table 23-22 describes the bit settings of AUTORSTSR.

**Table 23-22. AUTORSTSR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | RST_CKSTP | Core was reset in response to check stop<br>0 No reset<br>1 Reset occurred. |
| 1–3 | — | Reserved |
| 4 | RST_WRS | Core was reset in respone to watchdog timer expiration<br>0 No reset<br>1 Reset occurred |
| 5–7 | — | Reserved |
| 8 | RST_MPIC | Core was reset in respone to MPIC reset request<br>0 No reset<br>1 Reset occurred |
| 9–11 | — | Reserved |
| 12 | RST_CORE | Core was reset in respone to internal core request to reset itself by seeting bit DBCR[RST] register.<br>0 No reset<br>1 Reset occurred |
| 13–15 | — | Reserved |
| 16 | READY | Core ready pin. This bit reflects what is driven on the READY_P external signal.<br>0 Core not ready<br>1 Core ready |
| 17–19 | — | Reserved |
| 20 | RST_DPSLP | Core was reset in respone to deep sleep by seeting bit POWMGTCSR[DPSLP] register<br>0 No reset<br>1 Reset occurred |
| 21–31 | — | Reserved |

### 23.4.1.20 Processor Version Register (PVR)

Shown in Figure 23-20, the PVR contains the e500 processor version number. It is a memory-mapped copy of the PVR in the e500 core (and is therefore accessible to external devices). Section 5.3.1, “Processor Version Register (PVR) and System Version Register (SVR),” lists the complete values for the MPC8536E.

Offset 0x0A0 Access: Read only

| | 0 … 15 | 16 … 31 |
|---|---|---|
| R | Version | Revision |
| W | | |
| Reset | See Section 5.3.1, “Processor Version Register (PVR) and System Version Register (SVR) | |

**Figure 23-20. Processor Version Register (PVR)**

Table 23-23 describes the fields of PVR.

Table 23-23. PVR Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 32–47 | Version | A 16-bit number that identifies the version of the processor. Different version numbers indicate major differences between processors, such as which optional facilities and instructions are supported. (See Section 5.3.1, “Processor Version Register (PVR) and System Version Register (SVR),” for specific values.) |
| 48–63 | Revision | A 16-bit number that distinguishes between implementations of the version. Different revision numbers indicate minor differences between processors having the same version number, such as clock rate and engineering change level. (See Section 5.3.1, “Processor Version Register (PVR) and System Version Register (SVR),” for specific values.) |

### 23.4.1.21 System Version Register (SVR)

Shown in Figure 23-21, the SVR contains the system version number for the MPC8536E implementation. This value can also be read though the SVR SPR of the e500 core, described in the *PowerPC e500 Core Family Reference Manual*. Section 5.3.1, “Processor Version Register (PVR) and System Version Register (SVR),” lists the complete values for the MPC8536E.

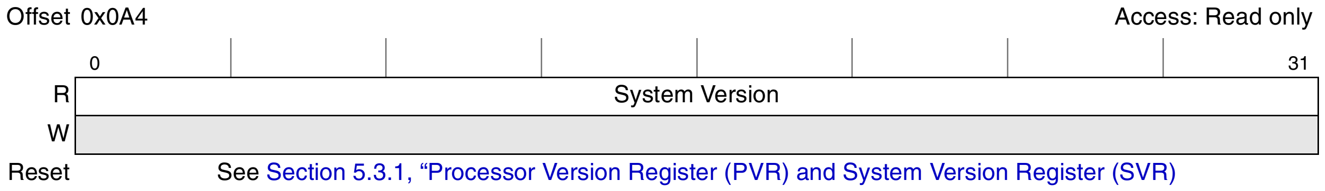

Offset 0x0A4
Access: Read only
0
31
R
System Version
W
Reset
See Section 5.3.1, “Processor Version Register (PVR) and System Version Register (SVR)

Figure 23-21. System Version Register (SVR)

Table 23-24 describes the fields of SVR.

Table 23-24. SVR Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–31 | SV | System version numbers for the MPC8536/MPC8536E system logic<br>0x803F_0091 for MPC8536E (with security)<br>0x8037_0091 for MPC8536 (without security). |

### 23.4.1.22 Reset Control Register (RSTCR)

Shown in Figure 23-22, the RSTCR contains the reset control bits.

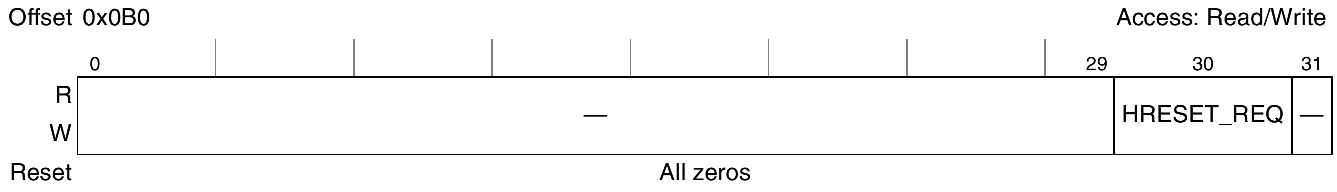


Figure 23-22. Reset Control Register (RSTCR)

Table 23-25 describes the bit settings of RSTCR.

Table 23-25. RSTCR Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–29 | — | Reserved |
| 30 | HRESET_REQ | Hardware reset request |
| 31 | — | Reserved |

### 23.4.1.23 LBC Voltage Select Control Register (LBCVSELCR)

Shown in Figure 23-23, the LBCVSELR contains local bus voltage control bits.

Offset 0x0C0 Access: Read/Write

| | 0–25 | 26–27 | 28–31 |
|---|---|---|---|
| R<br>W | — | LBCV | — |
| Reset | All zeros | | |

Figure 23-23. LBC Voltage Select Control Register (LBCVSELCR)

Table 23-26 describes the bit settings of LBCVSELCR.

Table 23-26. LBCVSELCR Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–25 | — | Reserved |
| 26–27 | LBCV | Selects the I/O voltage for the local bus<br>00 (default) 3.3V<br>01 2.5V<br>10 1.8V<br>11 reserved |
| 28–31 | — | Reserved |

### 23.4.1.24 DDR Clock Disable Register (DDRCLKDR)

Shown in Figure 23-24, the DDRCLKDR contains bits that allow disabling the clocks of the DDR SDRAM controller.

Offset 0xB28

| | 0–15 |
|---|---|
| R | — |
| W | |
| Reset | All zeros |

| | 16–25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|
| R | — | MCK0_DIS | MCK1_DIS | MCK2_DIS | MCK3_DIS | MCK4_DIS | MCK5_DIS |
| W | | | | | | | |
| Reset | All zeros | | | | | | |

**Figure 23-24. DDR Clock Disable Register (DDRCLKDR)**

Table 23-27 describes the bit settings of DDRCLKDR.

**Table 23-27. DDRCLKDR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–25 | — | Reserved |
| 26 | MCK0_DIS | DDR clock 0 disable<br>0 MCK0 is enabled.<br>1 MCK0 is disabled. |
| 27 | MCK1_DIS | DDR clock 1 disable<br>0 MCK1 is enabled.<br>1 MCK1 is disabled. |
| 28 | MCK2_DIS | DDR clock 2 disable<br>0 MCK2 is enabled.<br>1 MCK2 is disabled. |
| 29 | MCK3_DIS | DDR clock 3 disable<br>0 MCK3 is enabled.<br>1 MCK3 is disabled. |
| 30 | MCK4_DIS | DDR clock 4 disable<br>0 MCK4 is enabled.<br>1 MCK4 is disabled. |
| 31 | MCK5_DIS | DDR clock 5 disable<br>0 MCK5 is enabled.<br>1 MCK5 is disabled. |

### 23.4.1.25 Clock Out Control Register (CLKOCR)

Shown in Figure 23-25, the CLKOCR contains control bits that select the clock sources to be placed on the clock out (CLK_OUT) signal.

Offset 0xE00 Access: Read/Write

| | 0 | 1 … 25 | 26 … 31 |
|---|---|---|---|
| R<br>W | ENB | — | CLK_SEL |
| Reset | All zeros | | |

**Figure 23-25. Clock Out Control Register (CLKOCR)**

Table 23-28 describes the bit settings of CLKOCR.

**Table 23-28. CLKOCR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | ENB | Clock out enable<br>0 CLK_OUT signal is three-stated<br>1 CLK_OUT signal is driven according to CLKOCR[CLK_SEL] |
| 1–25 | — | Reserved |
| 26–31 | CLK_SEL | Clock out select<br>000000 CCB (platform) clock<br>000001 CCB (platform) clock divided by 2<br>000010 SYSCLK (echoes SYSCLK input)<br>000011 SYSCLK divided by 2 (demonstrates platform PLL lock)<br>10x010 PCI bus clock<br>10x011 PCI bus clock divided by 2<br>All other values are reserved |

### 23.4.1.26 ECM Control Register (ECMCR)

Shown in Figure 23-26, the ECMCR contains the uppermost bits of the SATA1, SATA2, USB1, USB2, USB3 and ESDHC address bus for all transaction initiated by these blocks.

Offset 0xE20 Read/Write

| | 0 … 3 | 4 … 7 | 8 … 11 | 12 … 15 |
|---|---|---|---|---|
| R<br>W | USB1_UPRADR | USB2_UPRADR | USB3_UPRADR | ESDHC_UPRADR |
| Reset | All zeros | | | |

| | 16 … 19 | 20 … 23 | 24 … 31 |
|---|---|---|---|
| R<br>W | SATA1_UPRADR | SATA2_UPRADR | — |
| Reset | All zeros | | |

**Figure 23-26. ECM Control Register (ECMCR)**

Table 23-29 describes the bit settings of ECMCR.

Table 23-29. ECMCR Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–3 | USB1_UPRADR | The uppermost bits of the USB1 address bus for all transactions initiated by the USB1 |
| 4–7 | USB2_UPRADR | The uppermost bits of the USB2 address bus for all transactions initiated by the USB2 |
| 8–11 | USB3_UPRADR | The uppermost bits of the USB3 address bus for all transactions initiated by the USB3 |
| 12–15 | ESDHC_UPRADR | The uppermost bits of the ESDHC address bus for all transactions initiated by the ESDHC |
| 16–19 | SATA1_UPRADR | The uppermost bits of the SATA1 address bus for all transactions initiated by the SATA1 |
| 20–23 | SATA2_UPRADR | The uppermost bits of the SATA2 address bus for all transactions initiated by the SATA2 |
| 24–31 | — | Reserved |

### 23.4.1.27 General Control Register (GCR)

Shown in Figure 23-27, GCR contains control bits used for pad control of deep sleep power-saving mode.

Offset 0xE60 Access: Read/Write

| | 0–9 | 10–11 | 12–20 | 21 | 22–31 |
|---|---|---|---|---|---|
| R<br>W | — | BYPASS | — | DEEPSLEEP_Z | — |
| Reset | 0 0 0 0 0 0 0 0 0 0 | 0 0 | 0 0 0 0 1 0 0 0 0 | 0 | 0 0 0 0 0 0 0 0 0 0 |

Figure 23-27. General Control Register (GCR)

Table 23-30 describes the bit settings of GCR.

Table 23-30. GCR Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–9 | — | Reserved |
| 10–11 | BYPASS | Bypass mode<br>Transactions eligible for bypassing other transactions are defined as follows:<br>00 Disable all bypassing (default).<br>01 Small read requests can bypass large requests from other on-chip network (OCN) ports.<br>10 Small read requests and write-with-response requests can bypass large requests from other on-chip network (OCN) ports.<br>11 All small requests can bypass large requests from other on-chip network (OCN) ports.<br>Rules for transaction bypassing:<br>• Maintain ordering for a given port—A transaction cannot bypass another from the same port.<br>• Identify potential “control requests”—Transactions must be 32 bytes of less to bypass another transaction.<br>• Guarantee forward progress—Once the subsequent entry in front of larger transaction in the queue has been replaced by a bypassing transaction, the “do not pass” bit of the queue entry of the larger transaction can no longer be bypassed. If the “do not pass” bit has been set for the immediate next entry in the queue following a small request (32 bytes or less), the “do not pass” bit of the small request is automatically set. |

**Table 23-30. GCR Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 12–20 | — | Reserved |
| 21 | DEEPSLEEP_Z | Deep sleep pad disable<br>0 Normal operation. In deep sleep all input and output pads remain driven as per normal functional operation, and inputs remain enabled.<br>1 When in deep sleep mode, output pads that are not used for wakeup events are tristated, and the receivers of pad inputs are disabled. When waking from Deep sleep, pad inputs are re-enabled as soon as the wakeup event occurs, but pad outputs are un-tristated only after the reset counter PMRCCR[RCNT] expires. This affects all digital I/O pins except the following:<br>• Dual eTSEC (including Ethernet management interface and GbE clocking but not 1588)<br>• Triple USB<br>• GPIO<br>• DDR<br>• Interrupts (IRQ[0:11], $\overline{\text{MCP}}$, $\overline{\text{UDE}}$, $\overline{\text{IRQ\_OUT}}$)<br>• System control ($\overline{\text{HRESET}}$, $\overline{\text{HRESET\_REQ}}$, $\overline{\text{SRESET}}$, $\overline{\text{CKSTP\_IN}}$, $\overline{\text{CKSTP\_OUT}}$)<br>• Debug (TRIG_IN, TRIG_OUT, MSRCID[0:4], MDVAL, CLK_OUT)<br>• Power management (ASLEEP, POWER_EN, POWER_OK)<br>• Clocking (SYSCLK, RTC, DDRCLK)<br>• DFT ($\overline{\text{LSSD\_MODE}}$, L1_TSTCLK, L2_TSTCLK, TEST_SEL) |
| 22–31 | — | Reserved |

## 23.4.1.28 SerDes1 Control Register 0 (SRDS1CR0)

Shown in Figure 23-28, SRDS1CR0 contains functional control bits for the SerDes1 logic.

Offset 0xE_3000 Access: Read/Write

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | | | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 16 | 17 | | 19 | 20 | 21 | | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | TXEQAD | | | — | TXEQEH | | | SDPD | — | | | | | | |
| Reset | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |

**Figure 23-28. SerDes1 Control Register 0 (SRDS1CR0)**

Table 23-31 describes the fields of SRDS1CR0.

**Table 23-31. SRDS1CR0 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–16 | — | Reserved |
| 17–19 | TXEQAD | Sets the peak value for output swing of transmitters and the amount of transmit equalization for lanes A–D.<br>Transmit equalization selection bus for lane A–D<br>000 No equalization<br>001 1.09x relative amplitude<br>010 1.2x relative amplitude<br>011 1.33x relative amplitude<br>100 1.5x relative amplitude<br>101 1.71x relative amplitude<br>110 2.0x relative amplitude<br>111 Reserved<br>Recommended setting per protocol:<br>PCI Express: 100 |
| 20 | — | Reserved |
| 21–23 | TXEQEH | Sets the peak value for output swing of transmitters and the amount of transmit equalization for lanes E–H<br>Transmit equalization selection bus for lanes E–H<br>000 No equalization<br>001 1.09x relative amplitude<br>010 1.2x relative amplitude<br>011 1.33x relative amplitude<br>100 1.5x relative amplitude<br>101 1.71x relative amplitude<br>110 2.0x relative amplitude<br>111 Reserved<br>Recommended setting per protocol:<br>PCI Express: 100 |
| 24 | SDPD | SerDes1 power down. This power down signal shuts down the PLL, all of the receiver amplifiers, all of the samplers and places the transmitters in 3-state.<br>0 Application mode<br>1 Block power down |
| 25–31 | — | Reserved |

### 23.4.1.29 SerDes1 Control Register 2 (SRDS1CR2)

Shown in Figure 23-29, SRDS1CR2 contains functional control bits for the SerDes1 logic. Individual lanes can be powered down using SRDSCR2[0:7]. It requires the entire SerDes1 to reset in order to activate a lane from powering down.

Offset 0xE_3008 Access: Read/Write

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | PDA | PDB | PDC | PDD | PDE | PDF | PDG | PDH | IPSEN | — | X3SA | X3SB | X3SC | X3SD | X3SE | X3SF |
| Reset | All zeros | | | | | | | | | | | | | | | |

| | 16 | 17 | 18 | 19 | | | | | | | | | | | | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | X3SG | X3SH | PPSEN | — | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

**Figure 23-29. SerDes1 Control Register 2 (SRD1SCR2)**

Table 23-32 describes the fields of SRDS1CR2.

**Table 23-32. SRDS1CR2 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | PDA | Lane A power down<br>0 Normal<br>1 Power down Lane A<br>Recommended setting per protocol:<br>PCI-Express: 0 |
| 1 | PDB | Lane B power down<br>0 Normal<br>1 Power down Lane B<br>Recommended setting per protocol:<br>PCI-Express: 0 |
| 2 | PDC | Lane C power down<br>0 Normal<br>1 Power down Lane C<br>Recommended setting per protocol:<br>PCI-Express: 0 |
| 3 | PDD | Lane D power down<br>0 Normal<br>1 Power down Lane D<br>Recommended setting per protocol:<br>PCI-Express: 0 |
| 4 | PDE | Lane E power down<br>0 Normal<br>1 Power down Lane E<br>Recommended setting per protocol:<br>PCI-Express: 0 |

Table 23-32. SRDS1CR2 Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 5 | PDF | Lane F power down<br>0 Normal<br>1 Power down Lane F<br>Recommended setting per protocol:<br>PCI-Express: 0 |
| 6 | PDG | Lane G power down<br>0 Normal<br>1 Power down Lane G<br>Recommended setting per protocol:<br>PCI-Express: 0 |
| 7 | PDH | Lane H power down<br>0 Normal<br>1 Power down Lane H<br>Recommended setting per protocol:<br>PCI-Express: 0 |
| 8 | IPSEN | Internal power save enable<br>0 SerDes1 power saving disabled<br>1 SerDes1 power saving enabled (recommended) |
| 9 | — | Reserved |
| 10 | X3SA | Lane A transmitter three-state<br>0 Normal<br>1 The transmitter ouput is disabled and place in a three-state condition<br>Recommended setting per protocol:<br>PCI-Express: 0 |
| 11 | X3SB | Lane B transmitter three-state<br>0 Normal<br>1 The transmitter ouput is disabled and place in a three-state condition<br>Recommended setting per protocol:<br>PCI-Express: 0 |
| 12 | X3SC | Lane C transmitter three-state<br>0 Normal<br>1 The transmitter ouput is disabled and place in a three-state condition<br>Recommended setting per protocol:<br>PCI-Express: 0 |
| 13 | X3SD | Lane D transmitter three-state<br>0 Normal<br>1 The transmitter ouput is disabled and place in a three-state condition<br>Recommended setting per protocol:<br>PCI-Express: 0 |
| 14 | X3SE | Lane E transmitter three-state<br>0 Normal<br>1 The transmitter ouput is disabled and place in a three-state condition<br>Recommended setting per protocol:<br>PCI-Express: 0 |

Table 23-32. SRDS1CR2 Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 15 | X3SF | Lane F transmitter three-state<br>0 Normal<br>1 The transmitter ouput is disabled and place in a three-state condition<br>Recommended setting per protocol:<br>PCI-Express: 0 |
| 16 | X3SG | Lane G transmitter three-state<br>0 Normal<br>1 The transmitter ouput is disabled and place in a three-state condition<br>Recommended setting per protocol:<br>PCI-Express: 0 |
| 17 | X3SH | Lane H transmitter three-state<br>0 Normal<br>1 The transmitter ouput is disabled and place in a three-state condition<br>Recommended setting per protocol:<br>PCI-Express: 0 |
| 18 | PPSEN | Pin power save enable<br>0 Pin power saving disabled<br>1 Pin power saving enabled (recommended) |
| 19–31 | — | Reserved |

## 23.4.1.30 SerDes2 Control Register 0 (SRDS2CR0)

Shown in Figure 23-30, the SRDS2CR0 contains the functional control bits for the SerDes2 logic.

Offset 0xE_3100 Access: Read/Write

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | | | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 16 | 17 | | 19 | 20 | 21 | | 23 | 24 | 25 | 26 | 27 | 28 | | | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | TXEQA | | | — | TXEQE | | | SDPD | — | | | | | | |
| Reset | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |

Figure 23-30. SerDes2 Control Register 0 (SRDS2CR0)

Table 23-33 describes the fields of SRDS2CR0.

**Table 23-33. SRDS2CR0 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–16 | — | Reserved |
| 17–19 | TXEQA | Sets the peak value for output swing of transmitters and the amount of transmit equalization for lane A. Transmit equalization selection bus for lane A.<br>If register field SRDSCR3[21:23] = 000, then the equalization definitions are:<br>000 No equalization<br>001 1.09x relative amplitude<br>010 1.2x relative amplitude<br>011 1.33x relative amplitude<br>100 1.5x relative amplitude<br>101 1.71x relative amplitude<br>110 2.0x relative amplitude<br>111 Reserved<br>If register field SRDSCR3[21:23]= 101, then the equalization definitions are:<br>000 No equalization<br>001 1.17x relative amplitude<br>010 1.4x relative amplitude<br>011 1.75x relative amplitude<br>100–111 Reserved<br>Recommended setting per protocol:<br>SGMII: 100<br>SATA: 001 |
| 20 | — | Reserved |
| 21–23 | TXEQE | Sets the peak value for output swing of transmitters and the amount of transmit equalization for lane E Transmit equalization selection bus for lane E.<br>If register field SRDSCR3[29:31] = 000, then the equalization definitions are:<br>000 No equalization<br>001 1.09x relative amplitude<br>010 1.2x relative amplitude<br>011 1.33x relative amplitude<br>100 1.5x relative amplitude<br>101 1.71x relative amplitude<br>110 2.0x relative amplitude<br>111 Reserved<br>If register field SRDSCR3[29:31]= 101, then the equalization definitions are:<br>000 No equalization<br>001 1.17x relative amplitude<br>010 1.4x relative amplitude<br>011 1.75x relative amplitude<br>100–111 Reserved<br>Recommended setting per protocol:<br>SGMII: 100<br>SATA: 001 |
| 24 | SDPD | SerDes2 power down. This power down signal shuts down the PLL, all of the receiver amplifiers, all of the samplers and places the transmitters in 3-state. For more information, refer to<br>0) Application mode<br>1) Block power down |
| 25–31 | — | Reserved |

### 23.4.1.31 SerDes2 Control Register 1 (SRDS2CR1)

Shown in Figure 23-31, the SRDS2CR1 contains the functional control bits for the SerDes2 logic. Individual lanes can be powered down using SRDS2CR1[0] and SRDS2CR1[4]. It requires the entire SerDes2 to reset in order to activate a lane from powering down.

Offset 0xE_3104 Access: Read/Write

| | 0 | 1 3 | 4 | 5 7 | 8 | 9 | 10 | 11 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | PDA | — | PDE | — | IPSEN | — | X3SA | — | X3SE | — |
| Reset | All zeros | | | | | | | | | |

| | 16 | 17 | 18 | 19 | 20 | 21 | | | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | | PPSEN | — | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

**Figure 23-31. SerDes2 Control Register 1 (SRDS2CR1)**

Table 23-34 describes the fields of SRDS2CR1.

**Table 23-34. SRDS2CR1 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | PDA | Lane A power down<br>0) Normal<br>1) Power down Lane A<br>Recommended setting per protocol:<br>SGMII: 0<br>SATA: 0 |
| 1–3 | — | Reserved |
| 4 | PDE | Lane E power down<br>0) Normal<br>1) Power down Lane E<br>Recommended setting per protocol:<br>SGMII: 0<br>SATA: 0 |
| 5–7 | — | Reserved |
| 8 | IPSEN | Internal power save enable<br>0 SerDes power saving disabled<br>1 SerDes power saving enabled (recommended) |
| 9 | — | Reserved |

**Table 23-34. SRDS2CR1 Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 10 | X3SA | Lane A transmitter three-state<br>0) Normal<br>1) The transmitter ouput is disabled and place in a three-state condition<br>Recommended setting per protocol:<br>SGMII: 0<br>SATA: 1 |
| 11–13 | — | Reserved |
| 14 | X3SE | Lane E transmitter three-state.<br>0) Normal<br>1) The transmitter ouput is disabled and place in a three-state condition<br>Recommended setting per protocol:<br>SGMII: 0<br>SATA: 1 |
| 15–17 | — | Reserved |
| 18 | PPSEN | Pin power save enable<br>0 Pin power saving disabled<br>1 Pin power saving enabled (recommended) |
| 19–31 | — | Reserved |

### 23.4.1.32 SerDes2 Control Register 2 (SRDS2CR2)

Show in Figure 23-32, the SRDS2CR2 contains the functional control bits used for the SerDes2 logic.

Offset 0xE_3108 Access: Read/Write

| | 0–18 | 19–23 | 24–26 | 27–31 |
|---|---|---|---|---|
| R<br>W | — | EICA | — | EICE |
| Reset | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 1 1 1 0 0 | 0 0 0 | 1 1 1 0 0 |

**Figure 23-32. SerDes2 Control Register 2 (SRDSCR2)**

Table 23-35 describes the fields of SRDS2CR2.

**Table 23-35. SRDS2CR2 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–18 | — | Reserved |
| 19–23 | EICA | SATA receiver electrical idle detection control for lane A.<br>Settings for bits 19–21:<br>000 Loss of signal detect function is disabled.<br>001 Default SGMII levels (low = 30 mV, high = 100 mV)<br>010 Intermediate level (low = 38 mV, high = 120 mV)<br>011 Intermediate level (low = 50 mV, high = 150 mV)<br>100 SATA1 levels (low = 65 mV, high = 175 mV)<br>101 Default SATA2 levels (low = 75 mV, high = 200 mV)<br>110 Intermediate level (low = 88 mV, high = 225 mV)<br>111 Intermediate level (low = 100mV, high = 250 mV)<br>Recommended setting per protocol:<br>• SGMII: 001<br>• SATA: 101<br>Settings for bits 22–23:<br>For SGMII:<br>00 Exit from Idle ~88UI and Unexpected Idle Detect ~1us (Application Mode)<br>01 Exit from Idle ~88UI and Unexpected Idle Detect ~10us<br>10 Exit from Idle ~48UI and Unexpected Idle Detect ~1us<br>11 Bypass<br>For SATA:<br>00 20 consecutive UI with no glitch (for exit from idle and for loss of signal detection).<br>01 40 consecutive UI with no glitch (for exit from idle and for loss of signal detection).<br>10 80 consecutive UI with no glitch (for exit from idle and for loss of signal detection).<br>11 20 consecutive UI with no glitch (for exit from idle and for loss of signal detection).<br>Recommended setting per protocol:<br>• SGMII: 00<br>• SATA: 00 |

**Table 23-35. SRDS2CR2 Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 24–26 | — | Reserved |
| 27–31 | EICE | SATA receiver electrical idle detection control for lane E.<br>Settings for bits 27–29:<br>000 Loss of signal detect function is disabled.<br>001 Default SGMII levels (low = 30 mV, high = 100 mV)<br>010 Intermediate level (low = 38 mV, high = 120 mV)<br>011 Intermediate level (low = 50 mV, high = 150 mV)<br>100 SATA1 levels (low = 65 mV, high = 175 mV)<br>101 Default SATA2 levels (low = 75 mV, high = 200 mV)<br>110 Intermediate level (low = 88 mV, high = 225 mV)<br>111 Intermediate level (low = 100mV, high = 250 mV)<br>Recommended setting per protocol:<br>• SGMII: 001<br>• SATA: 101<br>Settings for bits 30–31:<br>For SGMII:<br>00 Exit from Idle ~88UI and Unexpected Idle Detect ~1us (Application Mode)<br>01 Exit from Idle ~88UI and Unexpected Idle Detect ~10us<br>10 Exit from Idle ~48UI and Unexpected Idle Detect ~1us<br>11 Bypass<br>For SATA:<br>00 20 consecutive UI with no glitch (for exit from idle and for loss of signal detection).<br>01 40 consecutive UI with no glitch (for exit from idle and for loss of signal detection).<br>10 80 consecutive UI with no glitch (for exit from idle and for loss of signal detection).<br>11 20 consecutive UI with no glitch (for exit from idle and for loss of signal detection).<br>Recommended setting per protocol:<br>• SGMII: 00<br>• SATA: 00 |

### 23.4.1.33 SerDes2 Control Register 3 (SRDS2CR3)

Shown in Figure 23-33, the SRDS2CR3 contains the functional control bits for the SerDes2 logic.

Offset 0xE_310C Access: Read/Write

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | | KFRA | | KPHA | | SDFMA | | — | | KFRE | | KPHE | | SDFME | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

| | 16 | 17 | 18 | 23 | 24 | 25 | 26 | 31 |
|---|---|---|---|---|---|---|---|---|
| R<br>W | — | | SDTXLA | | — | | SDTXLE | |
| Reset | All zeros | | | | | | | |

**Figure 23-33. SerDes2 Control Register 3 (SRDS2CR3)**

Table 23-35 describes the fields of SRDS2CR3.

**Table 23-36. SRDS2CR3 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–1 | — | Reserved |
| 2–3 | KFRA | Selects the gain ’Kfr’ in the CDR for lane A.<br>00 $2^{-5}$<br>01 $2^{-6}$<br>10 Reserved<br>11 Reserved<br>Recommended setting per protocol:<br>• SGMII: N/A<br>• SATA: 01 |
| 4–5 | KPHA | Selects the gain ’Kph’ in the CDR for lane A.<br>00 Reserved<br>01 $2^{-7}$<br>10 $2^{-8}$<br>11 Reserved<br>Recommended setting per protocol:<br>• SGMII: N/A<br>• SATA: 01 |
| 6–7 | SDFMA | Sets the bandwidth of the digital filter to optimize for given frequency offset specification for lane A.<br>00 200 ppm (SGMII)<br>01 600 ppm (SATA)<br>10 Reserved<br>11 Reserved<br>Recommended setting per protocol:<br>• SGMII: 00<br>• SATA: 01 |
| 8–9 | — | Reserved |
| 10–11 | KFRE | Selects the gain ’Kfr’ in the CDR for lane E.<br>00 $2^{-5}$<br>01 $2^{-6}$<br>10 Reserved<br>11 Reserved<br>Recommended setting per protocol:<br>• SGMII: N/A<br>• SATA: 01 |
| 12–13 | KPHE | Selects the gain ’Kph’ in the CDR for lane E.<br>00 Reserved<br>01 $2^{–7}$<br>10 $2^{–8}$<br>11 Reserved<br>Recommended setting per protocol:<br>• SGMII: N/A<br>• SATA: 01 |

Table 23-36. SRDS2CR3 Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 14–15 | SDFME | Sets the bandwidth of the digital filter to optimize for given frequency offset specification for lane E.<br>00 200 ppm (SGMII)<br>01 600 ppm (SATA)<br>10 Reserved<br>11 Reserved<br>Recommended setting per protocol:<br>• SGMII: 00<br>• SATA: 01 |
| 16-17 | — | Reserved |
| 18–23 | SDTXLA | Controls lane A transmitter amplitude levels.<br>If SRDS2CR0[19] = 0, then Full Swing = Vdd/2 and bit settings are as follows:<br>bits [18–20] = Reserved<br>000 No amplitude reduction<br>001 0.916 × full swing<br>010 0.833 × full swing<br>011 0.750 × full swing<br>100 0.666 × full swing<br>101 0.583 × full swing<br>110 0.500 × full swing<br>111 Reserved<br>If SRDS2CR0[19] = 1, then Full Swing = 5/6 * Vdd/2 and bit settings are as follows:<br>bits [18–20] = Reserved<br>000 No amplitude reduction<br>001 0.916 × full swing<br>010 0.833 × full swing<br>011 0.750 × full swing<br>100 0.666 × full swing<br>101 0.583 × full swing<br>110 0.500 × full swing<br>111 Reserved<br>Recommended setting per protocol:<br>• SGMII: 000<br>• SATA: 101 |

Table 23-36. SRDS2CR3 Field Descriptions (continued)

| Bits | Name | Description |
|---|---|---|
| 24–25 | — | Reserved |
| 26–31 | SDTXLE | Controls lane E transmitter amplitude levels.<br>If SRDS2CR0[23] = 0, then Full Swing = Vdd/2 and bit settings are as follows:<br>bits [26–28] = Reserved<br>000 No amplitude reduction<br>001 0.916 × full swing<br>010 0.833 × full swing<br>011 0.750 × full swing<br>100 0.666 × full swing<br>101 0.583 × full swing<br>110 0.500 × full swing<br>111 Reserved<br>If SRDS2CR0[23] = 1, then Full Swing = 5/6 * Vdd/2 and bit settings are as follows:<br>bits [26–28] = Reserved<br>000 No amplitude reduction<br>001 0.916 × full swing<br>010 0.833 × full swing<br>011 0.750 × full swing<br>100 0.666 × full swing<br>101 0.583 × full swing<br>110 0.500 × full swing<br>111 Reserved<br>Recommended setting per protocol:<br>• SGMII: 000<br>• SATA: 101 |

## 23.5 Functional Description

This section describes the global utilities from a functional perspective.

### 23.5.1 Power Management Controller (PMC)

The PMC is responsible for maintaining the device in various low power modes.

#### 23.5.1.1 Overview

MPC8536E supports minimizing the power consumption at several levels.

- Dynamic power management
- Shutting down unused blocks
- Software controlled power-down state (doze, nap, sleep, deep sleep)

MPC8536E supports a deep sleep mode where power is removed to a portion of the die.

The PMC can gracefully stop the internal system bus and direct the memory controller to put DDR into self-refresh (if enabled).

In addition, the PMC controls the external power regulator switch to disable the VDD from a portion of the die.

The PMC allows several wake-up events source to exit low power mode, such as wake on LAN (magic packet or user defined), USB, GPIO, and internal timer. The wake-up events are mapped to OpenPIC interrupts to generate a wake-up interrupt to the PMC.

#### 23.5.1.2 Relationship Between Core and Device Power Management States

The MPC8536E has four low-power states: doze, nap, sleep and deep sleep. The mapping of core and device power management states is shown in Figure 23-34 showing state transitions from the perspective of the e500 core.

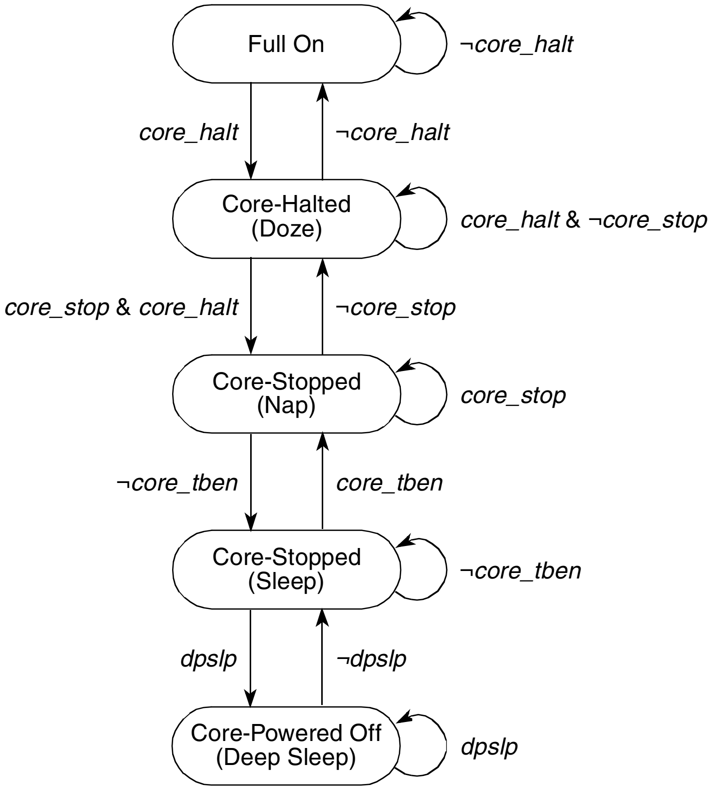


**Figure 23-34. e500 Core Power Management State Diagram**

For each operating state represented in the diagram, the cores state is listed first, with the corresponding state of the MPC8536E shown beneath it in parenthesis. Note that there are many other variables that control the state transitions between MPC8536E power management states. These additional variables are described in more detail in Section 23.5.1.8, “Power-Down Sequence Coordination.”

#### 23.5.1.3 CKSTP_IN0/1 is Not Power Management

CKSTP_IN0/1 are not described here because they are not considered power management signals, although asserting these do stop the cores and a stopped core is technically in a low-power mode. CKSTP_IN0/1 are described in Section 23.3.2, “Detailed Signal Descriptions.”

#### 23.5.1.4 Dynamic Power Management

Many blocks in the MPC8536E can dynamically turn off clocks within the block when sections of the block are idle. This feature is always enabled and occurs automatically.

#### 23.5.1.5 Shutting Down Unused Blocks

As described in Section 23.4.1.10, “Device Disable Register (DEVDISR),” DEVDISR provides a way to shut down certain functional blocks within the MPC8536E when they are not needed in a particular system. DEVDISR can be written by the e500 core or by an external master. Powering down a block in this way turns off all clocks to that block.

DEVDISR was designed with the expectation that, once initialized by software, it would be modified only by a hard system reset (HRESET). It is recommended that this register be written only during system initialization. Blocks disabled by DEVDISR must not be re-enabled without a hard reset. (Setting DEVDISR[TB] disables the core’s timer facilities, and setting DEVDISR[E500] places the core in the core_stopped state in which it does not respond to interrupts.) The results of re-enabling previously disabled blocks (by clearing the corresponding DEVDISR field) without a hard reset are undefined.

**NOTE**

Functional blocks disabled using DEVDISR cannot respond to configuration accesses. Any access to configuration, control, and status registers of a disabled block is a programming error.

#### 23.5.1.6 Software-Controlled Power-Down States

e500 software can place the device in doze, nap, or sleep power-down states by writing to HID0 in the core. In addition, external masters can write to the memory-mapped POWMGTCSR in the MPC8536E to cause the device to enter doze, sleep or deep sleep modes.

##### 23.5.1.6.1 Doze Mode

In doze mode, the e500 core suspends instruction execution, significantly reducing the power consumption of the core. Snooping of the L1 data cache is still supported and thus the data in the data cache is kept coherent. Interrupts directed to the core as described in are monitored by the device and cause the MPC8536E to use the defined handshake mechanism to exit the core from doze mode to allow the core to recognize and process the interrupt; however, unless the interrupt subroutine turns off (or masks) the control bits that enabled doze mode (MSR[WE], and HID0[DOZE]), the device re-enters doze mode after the interrupt has been serviced.

The e500 core’s timer facilities are still enabled during doze mode, and core time base interrupts can be generated. All device logic external to the core remains fully operational in doze mode. Additionally, ASLEEP and READY pins are both negated.

#### 23.5.1.6.2 Nap Mode

In nap mode all clocks internal to the e500 core are turned off except for its timer facilities clock (the core time base). The L1 caches do not respond to snoops in nap mode, so if coherency with external I/O transactions is required, the L1 cache must be flushed before entering nap mode.

Similar to doze mode, interrupts occurring in nap mode cause the device to wake up the e500 core in order to service the interrupt. However, unless the interrupt service routine changes the control bits that caused the device to enter nap mode (MSR[WE], and HID0[NAP]), the MPC8536E returns to nap mode after the interrupt is serviced.

All device logic external to the e500 core remains fully operational in nap mode. Additionally, ASLEEP and READY pins are both negated.

#### 23.5.1.6.3 Sleep Mode

In sleep mode, all clocks internal to the e500 core are turned off, including the timer facilities clock. Several modules clocks of the device logic are also shut down. Only the modules clocks which allows to wake up the MPC8536E are still running.

The modules which can be used as a wake up source are the Ethernet, USB controllers, GPIO, internal and external interrupts.

After the core and I/O interfaces have shut down, ASLEEP is asserted and READY is negated.

#### 23.5.1.6.4 Deep Sleep Mode

In deep sleep mode, all clocks internal to the e500 core are turned off, including the timer facilities clock. In addition the power supply is removed to the e500 core and the L2 cache.

Several modules’ clocks of the device logic are also shut down. Only those modules’ clocks which allow to wake up the MPC8536E are still running; modules which can be used as a wake up source are the Ethernet controllers, USB controllers, GPIO, and internal and external interrupts.

If the separate (asynchronous) PCI_CLK clock signal is used rather than SYSCLK as the PCI clock, then this clock must be constantly driven, even when in Deep Sleep mode, in order to avoid loss of lock.

For any SerDes that is not disabled through cfg_io_ports[0:2]=001 or cfg_srds2_prtcl[0:2]=111 respectively, the applicable SD_REF_CLK/$\overline{\text{SD\_REF\_CLK}}$ must be constantly driven, even when in Deep Sleep mode, in order to avoid loss of lock.

After the core and I/O interfaces have shut down, ASLEEP is asserted, READY is negated and POWER_EN is negated.

After the device is woken up by one of the wake-up events, the POWER_EN signal is asserted. The power management controller waits for POWER_OK indication from the regulator in order to make sure the power level is stable before enabling the the e500 core PLL. In case the POWER_OK is not driven from an external voltage regulator and is pullup high, the power managment controller will wait for the VRCNT (voltage ramp-up) timer to expire before enabling the e500 core PLL. The e500 core and the L2 cache are reset after the device exits from deep sleep mode.

When waking from Deep Sleep if GCR[DEEPSLEEP_Z]=1, pad inputs are re-enabled as soon as the wakeup event occurs, but pad outputs are un-tristated only after the reset counter PMRCCR[RCNT] expires.

#### 23.5.1.6.5 Jog Mode

Jog mode provides a dynamic mechanism to lower (or raise) the CPU core clock while leaving the platform clock rate unchanged ( for example, to optimize Tj (junction temperature) and power dissipation of the device). In doing this, the timing of an application at lower clock rate would behave lethargically, but all tasks and system timing would be maintained.

The term jog mode arose because it can be a slower version of run; however, despite the name of the mode, there is no requirement that it be used to slow down the core; it could equally be used to speed up the core (providing that the new core frequency is still within the frequency specifications of the MPC8536E). Jog mode does not impact static (leakage) power.

Before initiating jog mode, it is the user’s software responsibility to save state of the device as required. (The core is reset, but not the platform.) The user’s software must also configure the boot vector for the warm reset boot code as appropriate (similar to what is done for deep sleep mode). This typically involves modifying the boot page translation register (BPTR) and/or local access windows as required.

Peripherals in the platform need not be disabled by software; however, because they will not be operating during the jog mode frequency transition process it is possible that I/O peripherals such as PCI and eTSEC may lose packets during the jog mode frequency transition. Therefore, in certain applications the user may wish to disable the I/O peripherals manually before entering jog mode.

Note that as well as being used to define the new core frequency for a Jog Request, the PMJCR[e500_Ratio] is also used as the new e500 ratio when waking from Deep Sleep.

When a jog mode request is initiated (by setting POWMGTCSR[JOG]), the following sequence of events is performed:

1. The system operates as if a request to enter Sleep mode has occured, with the exception that the values written into the PMCDR register are ignored, and it is treated as if every bit in PMCDR is a logic 1. This means that the eTSECs, USB controllers, DDR and eLBC will be stopped.
2. The system isolates the outputs of the core complex (e500 and L2), as per Deep Sleep mode. However, power is NOT removed from the core complex and the POWER_EN output pin is not deasserted. Because power is NOT removed from the core complex, this also means that the inputs of the core complex are not isolated (unlike Deep Sleep).
3. Reset the core (warm reset) to initiate e500 boot. This is as per Deep Sleep mode. However, rather than setting the AUTORSTSR[rst_dpslp] bit, a different bit called AUTORSTSR[rst_jog] is set by hardware.
4. As per an exit from Deep Sleep, when the e500 PLL regains lock the platform clocks are re-enabled and the system resumes operation.
5. After a jog sequence as defined above, the new e500 clock ratio is reflected in PORPLLSR[e500_Ratio]. Therefore, after jog the values in PORPLLSR may not necessarily reflect the values driven by the the POR config pins.

**NOTE**

> The user must not issue a jog request at the same time as issuing a request for another low power mode, or while the system is in the process of entering a low power mode. The user also must not enter another low power mode when performing a jog sequence.
>
> In other words, POWMGTCSR[JOG] must never be set simultaneously with any other fields of POWMGTCSR.

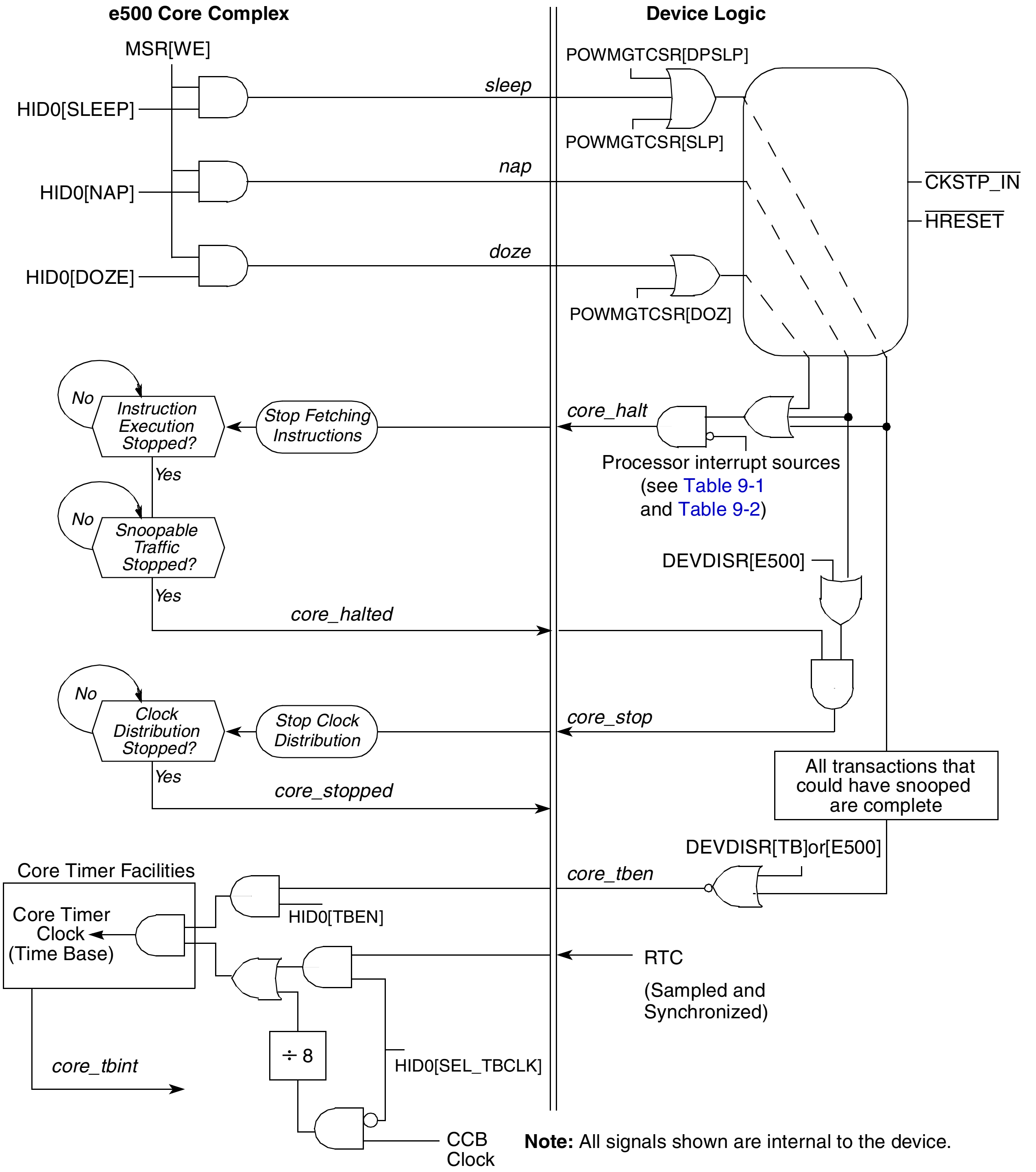
During the Jog mode process, ASLEEP is asserted and READY is negated. When the Jog sequence completes and the system resumes operation, ASLEEP is negated and READY is asserted.

#### 23.5.1.7 Power Management Control Fields (e500)

The e500 core provides the following fields to signal power management requests to the MPC8536E device logic:

- MSR[WE] Used to qualify the values of HID0[DOZE,NAP,SLEEP] in the generation of the internal doze, nap, and sleep signals.
- HID0[DOZE] Signals the MPC8536E to initiate doze mode.
- HID0[NAP] Signals the MPC8536E to initiate nap mode.
- HID0[SLEEP] Signals the MPC8536E to initiate sleep mode.

These register fields and their functional relationship are shown in Figure 23-35. The *PowerPC e500 Core Family Reference Manual* has details on accessing these power management control bits. An external master can also initiate power management requests by setting the DOZ, SLP or DPSLP bits in the memory-mapped power management control and status register (POWMGTCSR). Because the core responds to snoops while dozing but not while napping, maintaining cache coherency requires significant preparation by the core before entering nap mode. For this reason only the core can initiate a nap during normal operation while other masters can initiate a doze.



**Note:** All signals shown are internal to the device.

**Figure 23-35. MPC8536E Power Management Handshaking Signals**

### 23.5.1.8 Power-Down Sequence Coordination

To preserve cache coherency and otherwise avoid loss of system state, the core's transition to low-power modes is coordinated by a set of handshaking signals and protocols with all other MPC8536E functional blocks that respond to power-down requests. The mode-transition protocol is executes automatically under these conditions and is shown in Figure 23-35 and described in Table 23-37.

The column in Table 23-37 showing the global utilities block as initiating a low-power mode corresponds to the external masters that can write to the POWMGTCSR that resides in the global utilities block. For the MPC8536E, these are the PCI interfaces. However, note that the core can also write to POWMGTCSR and, in this case, can initiate power management through the global utilities block.

**Table 23-37. Power Management Entry Protocol and Initiating Functional Units**

| Low-Power Mode | Entry Protocol | Initiating Functional Unit | |
|---|---|---|---|
| | | Global Utilities | Core |
| Doze | 1. Assert *core_halt* input to core.<br>2. Wait for *core_halted* handshake from core.<br>3. Negate ASLEEP and READY | √ | √ |
| Nap | 1. Follow doze protocol<br>2. Assert *core_stop* input to core.<br>3. Wait for *core_stopped* handshake from core.<br>4. Negate ASLEEP and READY | — | √ |
| Sleep | 1. Follow doze protocol; send stop requests to rest of device.<br>2. Follow nap protocol.<br>3. Wait for all interfaces to acknowledge stop requests.<br>4. Assert ASLEEP, negate READY, power down all clocks except to PIC unit and units generating wakeup events. | √ | √ |
| Deep Sleep | 1. Follow doze protocol; send stop requests to rest of device.<br>2. Follow nap protocol.<br>3. Follow sleep protocol steps 1-3<br>4. Isolate inputs and outputs of core complex (e500 and L2)<br>5. Remove power to the core complex<br>6. Assert ASLEEP, negate READY, power down all clocks except to PIC unit and units generating wakeup events. | √ | |

As shown in Figure 23-35, the e500 core enters low-power modes only in response to the core_halt, core_stop, or core_tben inputs from the MPC8536Es power management logic. These inputs may be prompted by the core (by setting the NAP, DOZE, or SLEEP bits in the HID0 when enabled by setting MSR[WE]) or by an external master (by setting POWMGTCSR[DOZ,SLP,DPSLP].

Figure 23-35 shows how all the clocking to the core timer facilities is disabled by clearing HID0[TBEN]. When enabled, (HID0[TBEN] = 1), the clock source is either the CCB clock divided by eight (the default) or a synchronized version of the RTC input.

### 23.5.1.9 Interrupts and Power Management (e500)

Whether low-power modes are automatically re-enabled after an interrupt is processed differs depending on whether the low power mode was entered due to a write to the core MSR[WE] bit or the low power mode was entered due to a write to POWMGTCSR.

#### 23.5.1.9.1 Interrupts and Power Management Controlled by MSR[WE] (e500)

When an interrupt is asserted to the CPU, the core complex saves portions of the MSR to MCSRR1, CSRR1, or SRR1 (depending on the type of interrupt), and restores those values on return from the routine. MSR[WE], which gates the doze, nap, and sleep power management outputs (internal device signals) from the core complex, is always among the bits saved and restored; hence these outputs negate to the MPC8536E power management logic when the interrupt begins processing in the core. They return to their previous state when the core executes an **rfi**, **rfci**, or **rfmci** instruction.

NOTE

> Returning doze, nap, and sleep signals to their original state when MSR[WE] is restored differs from low power management is implemented on earlier PowerPC devices where MSR[POW], which enables power-down requests, is cleared when the processor exits a low-power state and is not automatically restored, as it is in Book E implementations.

#### 23.5.1.9.2 Interrupts and Power Management Controlled by POWMGTCSR (e500)

The IRQ_MSK and CI_MSK fields of the POWMGTCSR register prevent $\overline{int}$ interrupts or $\overline{cint}$ critical interrupts from waking the device from a low power state. This is true regardless of the method used to enter the low power state.

Any unmasked interrupt (not masked by the mask bits in the POWMGTCSR register) causes the POWMGTCSR[DOZ,SLP,DPSLP] fields to be cleared when it occurs. When such an interrupt occurs, the device returns to the normal operating mode and does not automatically attempt to return to a low power state after the interrupt is handled.

Note that interrupts caused by the unconditional debug event ($\overline{\text{UDE}}$) and machine check ($\overline{\text{MCP}}$) signals are not masked by the IRQ_MSK and CI_MSK fields; therefore, when these signals assert, the POWMGTCSR[DOZ,SLP,DPSLP] fields are cleared and the device will return to full power operation. See Section 23.4.1.12, “Power Management Control and Status Register (POWMGTCSR),” for detailed information about the bits of POWMGTCSR.

Note also that unmasked interrupts that occur while the device is in the process of going into the sleep state (before sleep is completely attained) can also cause the device to clear the POWMGTCSR[DOZ,SLP,DPSLP] fields and return the device to full power operation. In particular for deep sleep, this means that the setting of POWMGTCSR[DPSLP] does not guarantee that the core will be reset if an interrupt arrives in this situation.

### 23.5.1.10 Snooping in Power-Down Modes (e500)

When the MPC8536E is in doze mode, the e500 core is in the core-halted state and it snoops its L1 caches and full coherency is maintained. In deeper power-down modes, however, the e500 core does not respond to snoops. The MPC8536E does not perform dynamic bus snooping as described in the e500 Reference Manual. That is, when the e500 core is in the core-stopped state (which is the state of the core when the MPC8536E is in either the nap or sleep state), the core is not awakened to perform snoops on global transactions. Therefore, before entering nap, sleep or deep sleep modes, the L1 caches should be flushed if coherency is required during these power-down modes.

### 23.5.1.11 Software Considerations for Power Management (e500)

Setting MSR[WE] generates a request to the MPC8536E logic (external to the core complex) to enter a power saving state. It is assumed that the desired power-saving state (doze, nap, or sleep) was set up by setting the appropriate HID0 bit, typically at system start-up time. Setting WE has no direct effect on instruction execution, but is reflected on the internal doze, nap, and sleep signals, depending on the HID0 settings. To ensure a clean transition into and out of a power-saving mode, the following program sequence is recommended:

```
                  sync
                  mtmsr (WE)
                  isync
         loop:    br loop
```

### 23.5.1.12 Requirements for Reaching and Recovering from Sleep State

In order to successfully reach the sleep state, I/O traffic to the device must be stopped. The logic that controls the power down sequence waits for all I/O interfaces to become idle. In some applications this may happen eventually without actively shutting down interfaces, but most likely, software will have to take steps to shut down the eTSEC, PCI, PCI Express and USB device interfaces before issuing the command (either the write to the core MSR[WE] as described above or writing to POWMGTCSR) to put the device into sleep state.

The exception to this is that interfaces used for wake (USB or eTSEC) do not need to be shut down if they are the desired source of wake-up.

Prior to entering a sleep state, the SATA interface should be stopped with the following sequence:

1. Confirm that all commands are completed by checking Command Queue Register (CQR).
2. Write SControl[SPM] to 4'b0010 to initiate slumber mode power management. This will notify the device to go into slumber mode.
3. Poll SStatus[IPM] for 4'b0110 to confirm interface is in slumber mode power management.
4. Optional: Write HControl[HC_On] to 1'b0 to ask sataHost to go offline. This places the PHY in reset and saves additional power.
5. Optional: Poll HStatus[HS_On] for 1'b0 to confirm that sataHost is offline.

Upon exiting sleep mode, software should return these configuration bits to their normal state.

The PCI interfaces will begin retrying inbound transactions before entering a power down state. The PCI interfaces, however, could potentially be in an unknown state when they exit sleep if they were in the middle of a retry sequence when internal clocks were shut down. Therefore it is strongly recommended that system software clear the memory space bit in the PCI Bus Command Register before putting the device in sleep mode. Software may also need to set the Agent Config Lock bit of the PCI Bus Function Register so that the device will not respond to configuration transactions. Upon exiting sleep mode, software should return these configuration bits to their normal state.

As described in Section 23.5.1.10, “Snooping in Power-Down Modes (e500)”, the L1 caches should be flushed if coherency is required.

### 23.5.1.13 Requirements for Reaching and Recovering from Deep Sleep State

In order for the device to transition to the deep sleep state, POWMGTCSR[DPSLP] must be set. As part of this process, the system automatically transitions through the sleep state before entering deep sleep. Software will have to take steps to map the boot vector to the warm reset boot code and to program all necessary configuration and control registers (CCSRs), and to disable the L2 Cache (L2CTL[L2E] = 0) before issuing the command (writing to POWMGTCSR) to put the device into deep sleep. These steps are necessary in order for the core to successfully re-boot on wake-up and reset from deep sleep. After disabling the L2 Cache, either before or after deep sleep (but prior to re-enabling the L2 Cache), software must also flash invalidate the L2 Cache (L2CTL[L2I] = 1). In addition to this, the requirements for reaching and recovering from Sleep described above also must be met prior to entering deep sleep.

### 23.5.1.14 Requirements for Generating Wake-Up Events

The MPC8536E exits from low power modes based on a wake up interrupt from the OpenPIC. Any interrupt connected to the OpenPIC can be configured by the e500 software to generate a wake up interrupt.

#### 23.5.1.14.1 USB

When the wake up event is generated to the USB host, it could be from the following reasons:

- Power fault
- Disconnect
- Connect
- Remote Wakeup (resume signalling)

When the wake up event is generated to the USB device, it could be from the following reasons:

- Resume signalling (USB not idle)

Refer also to Section 21.6.4, “Suspend/Resume,” for more details on USB wake up events. The USB interrupt is connected to the OpenPIC to generate a wake up interrupt.

A USB interrupt can be generated either from the interrupt sources enabled by the USBINTR register, or from the wake-up interrupt enabled by the CONTROL[WU_INT_EN] register field. When using wake up from USB, software must clear the USBINTR register and set CONTROL[WU_INT_EN] to ensure that the USB will only generate an interrupt due to a valid wake up event.

Prior to entering sleep or deep sleep, software for the USB host controllers needs to ensure that they are idle by ensuring that USBCMD[ASE] = 0, USBCMD[PSE]=0, PORTSC[SUSP] = 1, and USBCMD[RS] = 0. Software should then wait until USBSTS[HCH] = 1 before placing the system in sleep or deep sleep modes. As described above, software also must clear the USBINTR register and set CONTROL[WU_INT_EN].

Prior to entering sleep or deep sleep, software for the USB device controller also needs to ensure that it is idle. Again, it must also must clear the USBINTR register and set CONTROL[WU_INT_EN].

When configured to wake on USB, the USB controller(s) interface to the off-chip USB PHY remains operational in sleep or deep sleep, but the USB Controller will not initiate any traffic to DDR. The user must set DDR_SDRAM_CFG[SREN] = 1 and optionally can also program DDR_SR_CNTR[SR_IT] to a non-zero value.

#### 23.5.1.14.2 GPIO

The GPIO wake up event occurs according to configuration by the e500 software to generate an interrupt.

The GPIO interrupt is connected to the OpenPIC to generate a wake up interrupt.

#### 23.5.1.14.3 Timer

The timer wake up event occurs according to configuration by the e500 software to generate an interrupt when the timer expires.

The timer interrupt is connected to the OpenPIC to generate a wake up interrupt.

The timer facilities are not available when the device is in sleep or deep sleep modes since either the clock to the e500 core will be gated off or the power to the e500 core will be removed.

#### 23.5.1.14.4 eTSEC Wake-on LAN—Magic Packet

The eTSEC supports two types of wake-up events:

- Magic Packet
- ARP (user-defined) Packet

Note that the eTSEC cannot supports both types of wake-up event simultaneously.

When wake-up on Magic Packet is desired, prior to entering sleep or deep sleep, the user should set the Magic packet enable bit in the Ethernet controller (MACCFG2[MPEN]) and clear the eTSEC clock disable bit in the PMCDR register (PMCDR[etsecX] = 0, where X = 1 or 3, for the eTSEC(s) that is being used for the magic packet). The Ethernet MAC blocks receives all traffic to the system and hunts for magic packet (ignoring all received frames except the magic packet). When a Magic packet is detected then the Magic packet enable bit is automatically cleared by the MAC hardware and set the wake up interrupt to the power management controller.

The user should configure the interrupt controller to enable the eTSEC error interrupt (which may be generated either when a Magic packet is received, or in various other error situations). It is the user’s responsibility to determine for their system which error interrupts should be masked, and which are critical errors that should be used as wakeup events.

Note that if the user configured the Ethernet MAC to wake-up on Magic packet but the MPC8536E exits low power mode by other wake-up event source, it is the user responsibility to clear the Magic packet enable bit, otherwise the Ethernet received traffic is blocked.

While in Magic Packet mode, the eTSEC will not initiate any traffic to DDR. The user should set DDR_SDRAM_CFG[SREN] = 1 and optionally can also program DDR_SR_CNTR[SR_IT] to a non-zero value.

When a Magic Packet is received, an interrupt is generated and the eTSEC hardware automatically clears MACCFG2[MPEN]. However, transactions after the magic packet continue to be dropped until the core wakes and the entire device comes out of its low-power mode. Therefore there is no chance of buffer overflow.

#### 23.5.1.14.5 eTSEC Wake-on-LAN—ARP (User Defined) Packet

The eTSEC can generate a wake-up event upon detecting an ARP (user defined) packet. Prior to entering sleep or deep sleep, the user needs to configure the wake-up packet header fields in the Ethernet controller parser. The eTSEC should also be configured to generate an interrupt when the ARP (user defined) packet is received.

Additionally, the user should clear the Magic packet enable bit in the Ethernet controller (MACCFG2[MPEN]) and the eTSEC clock disable bit in the PMCDR register (PMCDR[TSECx] = 0). The eTSEC generates an interrupt upon detecting the ARP packet only after the last RxBD packet is closed and data is stored in the external memory.

In general the user may wish to configure the interrupt controller to enable both the eTSEC receive interrupt (which will be generated when an ARP packet is received), and the eTSEC error interrupt (which may be generated in an error situation). It is the user’s responsibility to determine for their system which error interrupts should be masked, and which are critical errors that should be used as wakeup events.

While in this low-power mode, the eTSEC may continue to issue transactions to DDR. The user should set DDR_SR_CNTR[SR_IT] to a non-zero value, and ensure that DDR_SDRAM_CFG[SREN] = 1. This causes the DDR to enter self refresh mode after being idle for an user defined number of DDR clock cycles. Optionally, the user through software can decide keep the DDR always ON instead of in self refresh mode.

Note that using this method requires the user to configure the Ethernet controller filer to reject all kind of Ethernet frames beside the ARP (user-defined) packet; otherwise the DDR will exit self refresh although no ARP (user-defined) packet has been received.

After an ARP (user-defined) packet is received, the eTSEC remains in ARP (user-defined) packet filing mode, and packets of other types will be dropped until software changes the eTSEC filing rules to accept other packets for normal operation. Note that in the time between receiving the ARP (user-defined) packet, and the time when software re-enables the eTSEC normal operation, multiple ARP (user-defined) packets may be received.

### 23.5.1.15 External Power Supply Control

The following diagrams shows the assumed scenarios for controlling the power supply to the MPC8356E. An external source is required to switch off the CORE_VDD and CORE_AVDD supplies in low power mode. A commercial power switch can be used but there is usually a switching delay associated with these devices, sometimes around 1ms.

The target is to get an equal voltage on CORE_VDD/CORE_AVDD and PLAT_VDD/PLAT_AVDD supplies. It is done by using a FET transistor on the line of PLAT_VDD/PLAT_AVDD supplies that are“always on”. by this we get a same IR drop on CORE_VDD/CORE_AVDD and PLAT_VDD/PLAT_AVDD. The PMC is open/close the FET transistor by a power enable command so a voltage to the right blocks is given or not given.

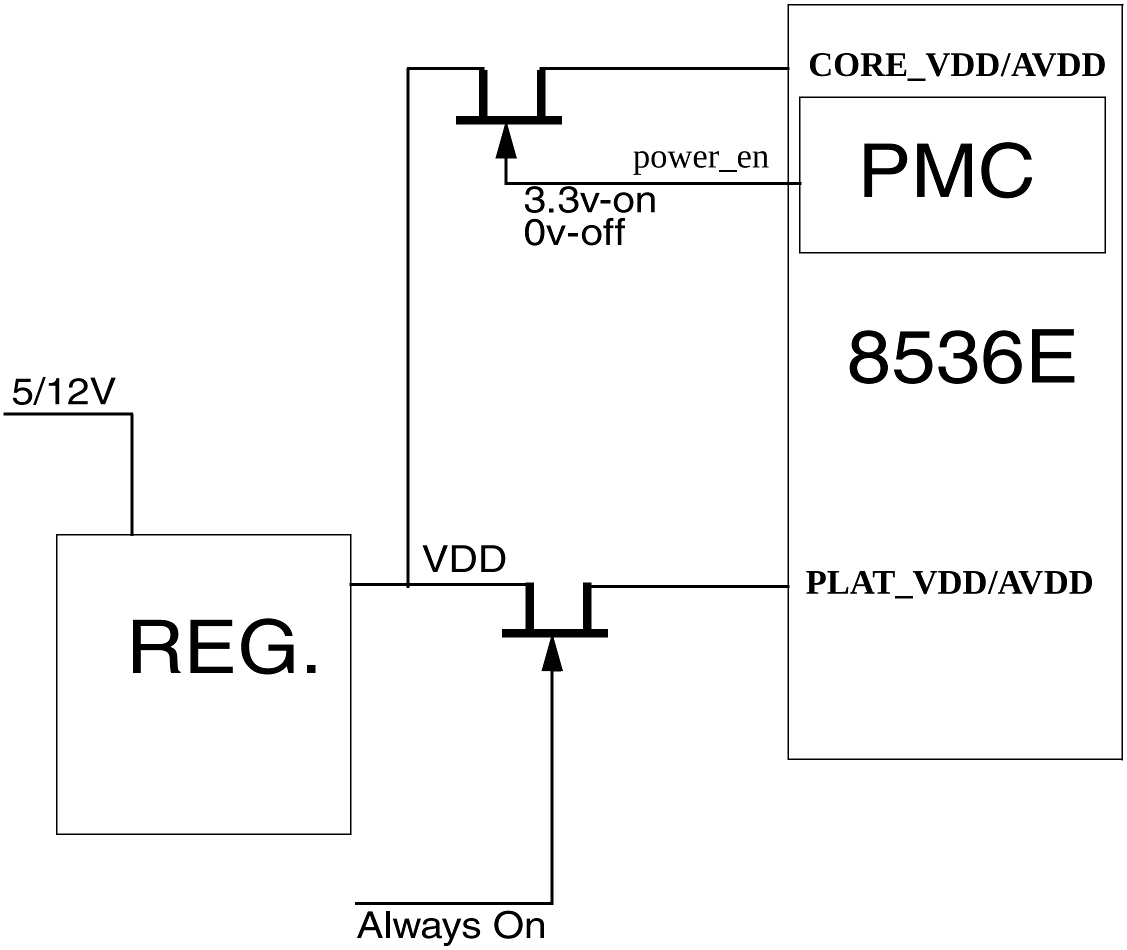


Figure 23-36. Power Supply Switch for MPC8536E

### 23.5.1.16 Low Power Considerations

The following should be considered in the low power system implementation.

#### 23.5.1.16.1 POWER_OK Input Signal

POWER_OK is an external input indication that VDD which was switched off in MPC8356E sleep mode has returned to specified levels after a wakeup event occurs. If an external power switch device is used (not a FET), it will typically provide such a signal. When a wakeup event occurs and PMC asserts the POWER_EN signal to turn on power, it will wait until the POWER_OK signal is asserted before it will proceed to enable the e500 PLL and to wait for the e500 PLL to lock. If there is no external source of POWER_OK, i.e. an external FET is used to switch power and there’s no way to indicate that power is stable, then the customer will tie POWER_OK to logic 1’b1 on the board. In this case POWER_OK will always be asserted to the PMC and the user will need to set the voltage ramp counter VRCNT in the PMRCCR register to ensure there is enough time for power to become stable before enabling the e500 PLL. If the e500 PLL is enabled before the power is stable it might become unpredictable and might not lock. Additionally, the user will need to set the reset counter RCNT in the PMRCCR register to ensure there is enough time for the e500 PLL to lock.

#### 23.5.1.16.2 POWER_EN Output Signal

POWER_EN signal is an external output from the power management controller that indicates to the external power regulator to toggle the power switch to on mode.

This signal is asserted when the system is in deep sleep mode and a wake-up event was accepted, but only after the counter of PMCCR[PDCNT] has finished counting. The signal deasserts after power is restored (POWER_OK is asserted and/or the voltage ramp-up counter VRCNT expires).

Assertion of POWER_EN signals the external power regulator to toggle the power switch on; its negation signals the regulator to toggle the power switch off. Assertion may occur only when a wakeup event occurs. Negation indicates no wakeup event occurs to the device.

The timing of POWER_EN is asynchronous; it is stable long enough so it is possible to synchronize it.

#### 23.5.1.16.3 DPSLP Register Bit

The POWMGTCSR[DPSLP] bit is set when the user wants to remove power to a portion of the die in deep sleep mode. This bit is cleared automatically by hardware upon receiving a wake up interrupt from the OpenPIC.

The Section 23.5.1.8, “Power-Down Sequence Coordination,” and Section 23.5.1.13, “Requirements for Reaching and Recovering from Deep Sleep State,” provide other important details on deep sleep mode.

#### 23.5.1.16.4 RST_DPSLP Register Bit

The AUTORSTSR[RST_DPSLP] bit is set when the core complex is reset in response to a deep sleep wake up event. This bit also allows boot code to distinguish between POR boot (cold reset) and boot from deep sleep (warm reset); this bit is cleared by the boot code. This register bit is referenced relative to the CCSRBAR register value. If the CCSRBAR register is modified from its default location (the MPC8536E configuration registers are moved to a different location in memory), boot software must take care to ensure it can still find the AUTORSTSR[RST_DPSLP] bit. It may be necessary before entering deep sleep to change the CCSRBAR register back to its default location.

# Chapter 24
# Device Performance Monitor

This chapter describes the device performance monitor facility, which can be used to monitor and optimize performance. The e500 core implements a separate performance monitor for strictly core-related behavior, such as instruction timing and L1 cache operations. This is described in the *PowerPC e500 Core Reference Manual* (Freescale Document Order No. E500CORERM).

Section 24.4.7, "Performance Monitor Events," briefly describes the events that can be monitored. Refer to the individual chapters for a better understanding of these events.

## 24.1 Introduction

The device-level performance monitor facility that can be used to monitor and record selected behaviors of the integrated device. Although the performance monitor described here is similar in many respects to the performance monitor facility implemented on the e500 core, it differs in that it is implemented using memory-mapped registers and it counts events outside the e500 core, for example, PCI, DDR, and L2 cache events.

Performance monitor counters (PMC0–PMC9) are used to count events selected by the performance monitor local control registers. PMC0 is a 64-bit counter specifically designated to count cycles. PMC1–PMC9 are 32-bit counters that can monitor 64 counter-specific events in addition to counting 64 reference events.

The benefits of the on-chip performance monitor are numerous, and include the following:

- Because some systems or software environments are not easily characterized by signal traces or benchmarks, the performance monitor can be used to understand the device's behavior in any system or software environment.
- The performance monitor facility can be used to aid system developers when bringing up and debugging systems.
- System performance can be increased by monitoring memory hierarchy behavior. This can help to optimize algorithms used to schedule or partition tasks and to refine the data structures and distribution used by each task.

### 24.1.1 Overview

Figure 24-1 is a high-level block diagram of the performance monitor, which consists of a global control register (PMGC0), one 64-bit counter (PMC0), nine 32-bit counters, and two control registers per counter (20 total control registers). The global control register PMGC0 affects all counters and takes priority over local control registers. The local control registers are divided into two groups, as follows:

- Local control A registers control counter freezing, overflow condition enable, event selection, and burstiness. Local control register PMLCA0, which controls counter PMC0, does not contain event selection because PMC0 counts only cycles.
- Local control B registers control the start and stop triggering, contain the counters' threshold values, and the value of the threshold multiplier. Local control register PMLCB0, which controls PMC0, does not contain threshold information because PMC0 only counts cycles.

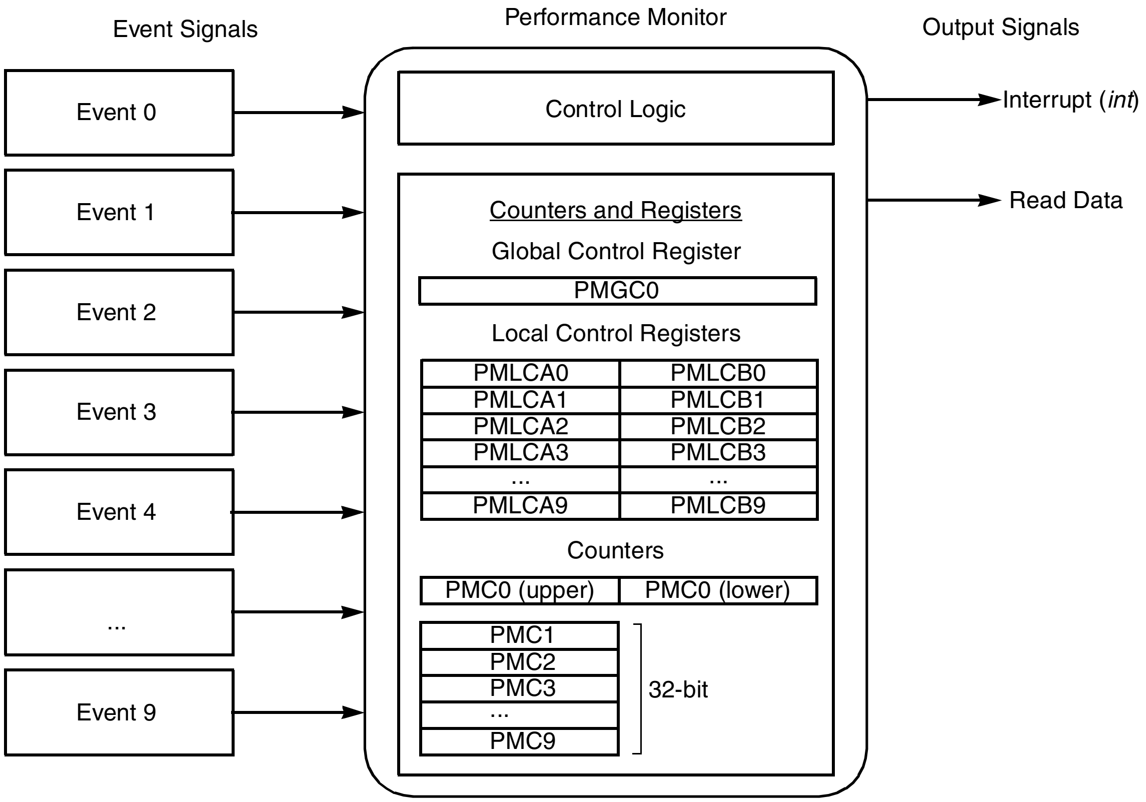


**Figure 24-1. Performance Monitor Block Diagram**

Performance monitor events are signalled by the functional blocks in the integrated device and are selectively recorded in the PMCs. Sixty-four of these events are referred to as reference events, which can be counted on any of the nine 32-bit counters. Counter-specific events can be counted only on the counter where the event is defined.

The performance monitor can generate an interrupt on overflow. Several control registers specify how a performance monitor interrupt is signalled. The PMCs can also be programmed to freeze when an interrupt is signalled.

### 24.1.2 Features

The performance monitor offers a rich set of features that permits a complete performance characterization of the implementation. These features include:

- One 64-bit counter exclusively dedicated to counting cycles
- Nine 32-bit counters that count the occurrence of selected events
- One global control register (affects all counters) and two local control registers per counter
- Ability to count up to 64 reference events that may be counted on any of the nine 32-bit counters
- Ability to count up to 576 counter-specific events
- Triggering and chaining capability
- Duration and quantity threshold counting
- Burstiness feature that permits counting of burst events with a programmable time between bursts
- Ability to generate an interrupt on overflow

## 24.2 Signal Descriptions

The performance monitor does not have any signals that are driven externally (off-chip) but it does assert the internal interrupt (*int*) signal on a performance monitor interrupt condition.

## 24.3 Memory Map and Register Definition

Performance monitor registers reside in the run-time register block starting at offset 0xE_1000. Undefined 4-byte address spaces within offset 0x000–0xFFF are reserved. This section describes the registers implemented to support the performance monitor facilities. Table 24-1 lists the performance monitor registers. These registers can be read or written only with 32-bit accesses.

### 24.3.1 Register Summary

The performance monitor uses ten counter registers and a group of local control registers that are used to specify the method of counting. Two local control registers are associated with each counter in addition to a global control register that applies to all counters.

In this table and in the register figures and field descriptions, the following access definitions apply:

- Reserved fields are always ignored for the purposes of determining access type.
- R/W, R, and W (read/write, read only, and write only) indicate that all the non-reserved fields in a register have the same access type.
- w1c indicates that all of the non-reserved fields in a register are cleared by writing ones to them.
- Mixed indicates a combination of access types.
- Special is used when no other category applies. In this case the register figure and field description table should be read carefully.

Table 24-1. Control Register Memory Map

| Address Offset (in Hex) | Register | Access | Reset | Section/Page |
|---|---|---|---|---|
| 0xE_1000 | PMGC0—Performance monitor global control register | R/W | 0x0000_0000 | 24.3.2.1/24-5 |
| 0xE_1010 | PMLCA0—Performance monitor local control register A0 | R/W | 0x0000_0000 | 24.3.2.2/24-6 |
| 0xE_1014 | PMLCB0—Performance monitor local control register B0 | R/W | 0x0000_0000 | 24.3.2.2/24-6 |
| 0xE_1018 | PMC0 (lower)—Performance monitor counter 0 lower | R/W | 0x0000_0000 | 24.3.3.1/24-10 |
| 0xE_101C | PMC0 (upper)—Performance monitor counter 0 upper | R/W | 0x0000_0000 | 24.3.3.1/24-10 |
| 0xE_1020 | PMLCA1—Performance monitor local control register A1 | R/W | 0x0000_0000 | 24.3.2.2/24-6 |
| 0xE_1024 | PMLCB1—Performance monitor local control register B1 | R/W | 0x0000_0000 | 24.3.2.2/24-6 |
| 0xE_1028 | PMC1—Performance monitor counter 1 | R/W | 0x0000_0000 | 24.3.3.1/24-10 |
| 0xE_1030 | PMLCA2—Performance monitor local control register A2 | R/W | 0x0000_0000 | 24.3.2.2/24-6 |
| 0xE_1034 | PMLCB2—Performance monitor local control register B 2 | R/W | 0x0000_0000 | 24.3.2.2/24-6 |
| 0xE_1038 | PMC2—Performance monitor counter 2 | R/W | 0x0000_0000 | 24.3.3.1/24-10 |
| 0xE_1040 | PMLCA3—Performance monitor local control register A3 | R/W | 0x0000_0000 | 24.3.2.2/24-6 |
| 0xE_1044 | PMLCB3—Performance monitor local control register B3 | R/W | 0x0000_0000 | 24.3.2.2/24-6 |
| 0xE_1048 | PMC3—Performance monitor counter 3 | R/W | 0x0000_0000 | 24.3.3.1/24-10 |
| 0xE_1050 | PMLCA4—Performance monitor local control register A4 | R/W | 0x0000_0000 | 24.3.2.2/24-6 |
| 0xE_1054 | PMLCB4—Performance monitor local control register B4 | R/W | 0x0000_0000 | 24.3.2.2/24-6 |
| 0xE_1058 | PMC4—Performance monitor counter 4 | R/W | 0x0000_0000 | 24.3.3.1/24-10 |
| 0xE_1060 | PMLCA5—Performance monitor local control register A5 | R/W | 0x0000_0000 | 24.3.2.2/24-6 |
| 0xE_1064 | PMLCB5—Performance monitor local control register B 5 | R/W | 0x0000_0000 | 24.3.2.2/24-6 |
| 0xE_1068 | PMC5—Performance monitor counter 5 | R/W | 0x0000_0000 | 24.3.3.1/24-10 |
| 0xE_1070 | PMLCA6—Performance monitor local control register A6 | R/W | 0x0000_0000 | 24.3.3.1/24-10 |
| 0xE_1074 | PMLCB6—Performance monitor local control register B6 | R/W | 0x0000_0000 | 24.3.2.2/24-6 |
| 0xE_1078 | PMC6—Performance monitor counter 6 | R/W | 0x0000_0000 | 24.3.3.1/24-10 |
| 0xE_1080 | PMLCA7—Performance monitor local control register A7 | R/W | 0x0000_0000 | 24.3.2.2/24-6 |
| 0xE_1084 | PMLCB7—Performance monitor local control register B7 | R/W | 0x0000_0000 | 24.3.2.2/24-6 |
| 0xE_1088 | PMC7—Performance monitor counter 7 | R/W | 0x0000_0000 | 24.3.3.1/24-10 |
| 0xE_1090 | PMLCA8—Performance monitor local control register A8 | R/W | 0x0000_0000 | 24.3.2.2/24-6 |
| 0xE_1094 | PMLCB8—Performance monitor local control register B8 | R/W | 0x0000_0000 | 24.3.2.2/24-6 |
| 0xE_1098 | PMC8—Performance monitor counter 8 | R/W | 0x0000_0000 | 24.3.3.1/24-10 |
| 0xE_10A0 | PMLCA9—Performance monitor local control register A9 | R/W | 0x0000_0000 | 24.3.2.2/24-6 |

Table 24-1. Control Register Memory Map (continued)

| Address Offset (in Hex) | Register | Access | Reset | Section/Page |
|---|---|---|---|---|
| 0xE_10A4 | PMLCB9—Performance monitor local control register B9 | R/W | 0x0000_0000 | 24.3.2.2/24-6 |
| 0xE_10A8 | PMC9—Performance monitor counter 9 | R/W | 0x0000_0000 | 24.3.3.1/24-10 |

In addition to these registers, the interrupt control provides four pairs of mask registers that can be used to monitor message, interprocessor, timer, and external interrupts. See Section 9.3.4, “Performance Monitor Mask Registers (PMMRs),” on page 9-32.

## 24.3.2 Control Registers

This section describes the performance monitor control registers in detail.

### 24.3.2.1 Performance Monitor Global Control Register (PMGC0)

The performance monitor global control register (PMGC0), shown in Figure 24-2, is a 32-bit register used to control all PMCs.

Offset 0xE_1000 Access: Read/Write

| | 0 | 1 | 2 | 3 … 31 |
|---|---|---|---|---|
| R<br>W | FAC | PMIE | FCECE | — |
| Reset | All zeros | | | |

Figure 24-2. Performance Monitor Global Control Register (PMGC0)

Table 24-2 describes PMGC0 fields.

Table 24-2. PMGC0 Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0 | FAC | Freeze all counters.<br>0 PMCs are incremented (if permitted by other PMGC0/PMLC bits).<br>1 PMCs are not incremented. Set by hardware when an interrupt is signalled and FCECE =1. |
| 1 | PMIE | Performance monitor interrupt enable. Interrupts are caused by PMC overflows.<br>0 Interrupts are disabled.<br>1 Interrupts are enabled and occur when an enabled condition or event occurs. |
| 2 | FCECE (DISCOUNT) | Freeze counters on enabled condition or event. An enabled condition or event is defined as:<br>The msb = 1 in PMC*n* and PMLCA*n*[CE] = 1.<br>The use of the trigger and freeze counter conditions depends on the enabled condition.<br>0 PMCs can be incremented (if permitted by other control bits).<br>1 PMCs can be incremented (if permitted by other control bits) only until an enabled condition or event occurs, at which time PMGC0[FAC] is set. It is up to software to clear FAC. |
| 3–31 | — | Reserved |

### 24.3.2.2 Performance Monitor Local Control Registers (PMLCA*n*, PMLCB*n*)

The performance monitor local control registers (PMLCA*n* and PMLCB*n*) are used to control the operation of the PMCs. The performance monitor local control A and B registers are paired 32-bit control registers that are associated with an individual counter to specify how the counter is used and what event is monitored on that counter.

Figure 24-3 shows the performance monitor local control A0 register (PMLCA0).

Offset 0xE_1010 Access: Read/Write

| | 0 | 1–4 | 5 | 6–31 |
|---|---|---|---|---|
| R<br>W | FC | — | CE | — |
| Reset | All zeros | | | |

**Figure 24-3. Performance Monitor Local Control Register A0 (PMLCA0)**

Table 24-3 describes PMLCA0 fields.

**Table 24-3. PMLCA0 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | FC | Freeze counter. Basic counter enable.<br>0 The PMCs are enabled and incremented (if permitted by other SPM control bits).<br>1 The PMCs are disabled–they do not increment. |
| 1–4 | — | Reserved |
| 5 | CE | Condition enable. Controls counter overflow condition. Should be cleared when PMC0 is used as a trigger or is selected for chaining.<br>0 Overflow conditions for PMC0 cannot occur (PMC0 cannot cause interrupts or freeze counters)<br>1 Overflow conditions occur when PMC0[msb] is set. |
| 6–31 | — | Reserved |

Figure 24-4 shows the performance monitor local control registers A1–A9.

Offset 0xE_1020
0xE_1030
0xE_1040
0xE_1050
0xE_1060
0xE_1070
0xE_1080
0xE_1090
0xE_10A0
0xE_10B0
0xE_10C0

Access: Read/Write

| | 0 | 1 4 | 5 | 6 8 | 9 15 | 16 20 | 21 25 | 26 31 |
|---|---|---|---|---|---|---|---|---|
| R<br>W | FC | — | CE | — | EVENT | BSIZE | BGRAN | BDIST |
| Reset | All zeros | | | | | | | |

**Figure 24-4. Performance Monitor Local Control A Registers (PMLCA1–PMLCA9)**

Table 24-4 describes PMLCA*n* fields.

**Table 24-4. PMLCA1–PMLCA9 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | FC | Freeze counter<br>0 The PMCs are incremented (if permitted by other PMC control bits).<br>1 The PMCs are not incremented (if permitted by other PMC control bits). |
| 1–4 | — | Reserved |
| 5 | CE | Condition enable<br>0 Overflow conditions for PMC*n* cannot occur (PMC*n* cannot cause interrupts or freeze counters). Should be cleared when PMC*n* is used as a trigger or is selected for chaining.<br>1 Overflow conditions occur when PMC*n*[msb] is set. |
| 6–8 | — | Reserved |
| 9–15 | EVENT | Event selector. Up to 128 events selectable.<br>Note that with counter-specific events, an offset of 64 must be used when programming the field, because counter-specific events occupy the bottom 64 values of the 7-bit event field where events are numbered. For example, to specify counter-specific event 0, the event field must be programmed to 64.<br>See Table 24-10 for definition of events. |
| 16–20 | BSIZE | Burst size. Fewest event occurrences that constitute a burst, that is, a rapid sequence of events followed by a relatively long pause. A value less than two implies regular event counting. Any non-threshold, regular event may be counted in a bursty fashion. See Section 24.4.6, “Burstiness Counting,” for more information. |
| 21–25 | BGRAN | Burst granularity. The maximum number of clock cycles between events that are considered part of a single burst. See Section 24.4.6, “Burstiness Counting.” |
| 26–31 | BDIST | Burst distance (used with TBMULT). The number of clock cycles between bursts. Must be set to a value greater than BSIZE for proper burstiness counting behavior.<br>00_0000 Regular counting |

Figure 24-5 shows the performance monitor local control B0 register (PMLCB0).

Offset 0xE_1014 Access: Read/Write

| | 0 1 | 2 5 | 6 7 | 8 11 | 12 13 | 14 15 | 16 31 |
|---|---|---|---|---|---|---|---|
| R<br>W | — | TRIGONSEL | — | TRIGOFFSEL | TRIGONCNTL | TRIGOFFCNTL | — |
| Reset | All zeros | | | | | | |

**Figure 24-5. Performance Monitor Local Control Register B0 (PMLCB0)**

Table 24-5 describes PMLCB0 fields.

**Table 24-5. PMLCB0 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–1 | — | Reserved |
| 2–5 | TRIGONSEL | Trigger-on select. The number of the counter that starts event counting. When the specified counter's TRIGONCNTL event overflows, the current counter begins counting. No triggering occurs if the value is self-referential, that is, when set to the current counter number. |
| 6–7 | — | Reserved |
| 8–11 | TRIGOFFSEL | Trigger-off select. The number of the counter that stops event counting. When the specified counter's TRIGONCNTL event overflows, the current counter stops counting. No triggering occurs if the value is self-referential, that is, when set to the current counter number. |
| 12–13 | TRIGONCNTL | Trigger-on control. Indicates the condition under which triggering to start counting occurs<br>00 Trigger off (no triggering to start)<br>01 Trigger on change<br>10 Trigger on overflow<br>11 Reserved |
| 14–15 | TRIGOFFCNTL | Trigger-off control. Indicates the condition under which triggering to stop occurs<br>00 Trigger off (no triggering to stop)<br>01 Trigger on change<br>10 Trigger on overflow<br>11 Reserved |
| 16–31 | — | Reserved |

Figure 24-6 shows performance monitor local control registers 1–9.

Offset 0xE_1024
0xE_1034
0xE_1044
0xE_1054
0xE_1064
0xE_1074
0xE_1084
0xE_1094
0xE_10A4
0xE_10B4
0xE_10C4

Access: Read/Write

| | 0 1 | 2 5 | 6 7 | 8 11 | 12 13 | 14 15 | 16 20 | 21 23 | 24 25 | 26 31 |
|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | TRIGONSEL | — | TRIGOFFSEL | TRIGONCNTL | TRIGOFFCNTL | — | TBMULT | — | THRESHOLD |
| Reset | All zeros | | | | | | | | | |

**Figure 24-6. Performance Monitor Local Control Register B (PMLCB1–PMLCB9)**

Table 24-6 describes PMLCB*n* fields.

**Table 24-6. PMLCB*n* Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–1 | — | Reserved |
| 2–5 | TRIGONSEL | Trigger-on select. Set this field equal to the number of the counter that should trigger event counting to start. When the specified counter's TRIGONCNTL event overflows, the current counter begins counting. No triggering occurs when TRIGONSEL = current counter. |
| 6–7 | — | Reserved |
| 8–11 | TRIGOFFSEL | Trigger-off select. Set this field equal to the number of the counter that should trigger event counting to stop. When the specified counter's TRIGONCNTL event overflows, the current counter stops counting. No triggering occurs when TRIGOFFSEL = current counter. |
| 12–13 | TRIGONCNTL | Trigger-on control. Indicates the condition under which triggering to start counting occurs<br>00 Trigger off (no triggering to start)<br>01 Trigger on change<br>10 Trigger on overflow<br>11 Reserved |
| 14–15 | TRIGOFFCNTL | Trigger-off control. Indicates the condition under which triggering to stop occurs<br>00 Trigger off (no triggering to stop)<br>01 Trigger on change<br>10 Trigger on overflow<br>11 Reserved |
| 16–20 | — | Reserved |

**Table 24-6. PMLCB*n* Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 21–23 | TBMULT | Threshold and burstiness multiplier. Threshold events are counted when the event duration exceeds a specified threshold value. The threshold is scaled based on the TBMULT settings.TBMULT is not used to scale the threshold value for quantity threshold events. The burst distance for burstiness counting is also scaled using the TBMULT settings. For all events that scale the threshold, the threshold field is multiplied by the factors shown below (ranging from 1 to 128).<br>000 1<br>001 2<br>010 4<br>011 8<br>100 16<br>101 32<br>110 64<br>111 128 |
| 24–25 | — | Reserved |
| 26–31 | THRESHOLD | Threshold. Only events whose (number of) occurrences exceed this value are counted. By varying the threshold value, software can characterize the events subject to the threshold. For example, if PMC2 counts eTSEC BD read latencies for which the duration exceeds the threshold, software can obtain the distribution of eTSEC BD read latencies for a given program by monitoring the program using various threshold values. |

## 24.3.3 Counter Registers

This section describes the PMCs in detail.

**NOTE**

> Because accessing a PMC manually has priority over incrementing it due to event counting, writing a PMC while it is counting may affect the count. Likewise, writing a performance monitor control register while its target counter is counting may also affect the count.

### 24.3.3.1 Performance Monitor Counters (PMC0–PMC9)

PMC0–PMC9 are used to count events selected by the performance monitor local control registers. PMC0, shown in Figure 24-7, is associated with two 32-bit registers that form a 64-bit counter designated to count clock cycles. PMC0 upper represents the upper 32 bits of counter 0, and PMC0 lower represents the lower 32 bits.

Offset 0xE_101C 0xE_1018 Access: Read/Write

| | 0–31 | 32–63 |
|---|---|---|
| R<br>W | PMC0 upper | PMC0 lower |
| Reset | All zeros | |

**Figure 24-7. Performance Monitor Counter Register 0 (PMC0)**

Table 24-7 describes PMC0 fields.

**Table 24-7. PMC0 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–63 | PMC0 | Event count. Counts only clock cycles |

PMC1–PMC9, shown in Figure 24-8, are 32-bit counters that can monitor 64 unique events in addition to the 64 reference events that can be counted on all of these registers.

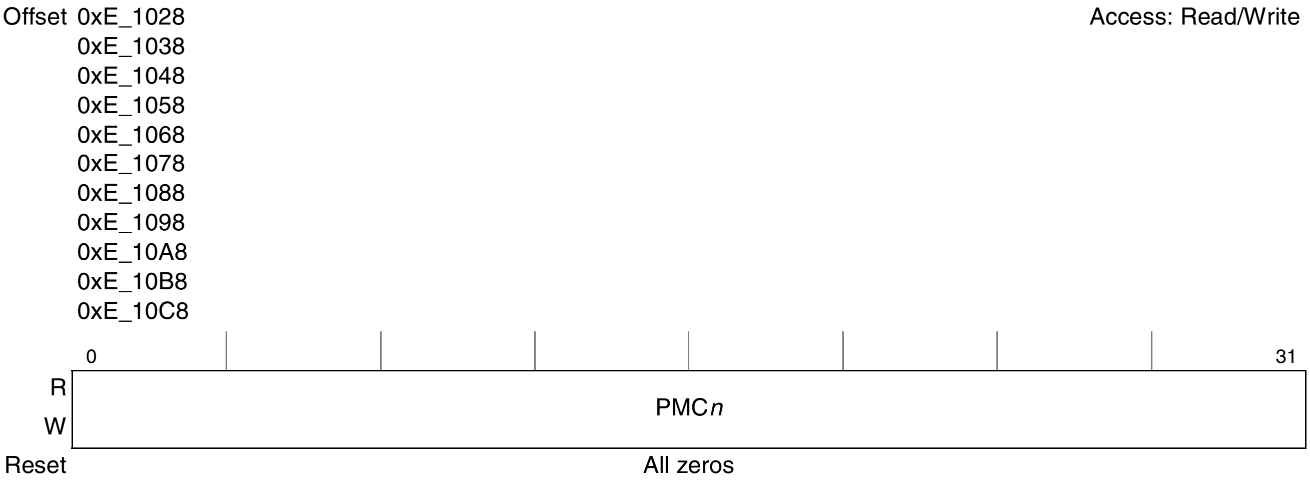


**Figure 24-8. Performance Monitor Counter Register (PMC1–PMC9)**

Table 24-8 describes PMC*n* fields.

**Table 24-8. PMC[1–9] Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–31 | PMC*n* | Event count. An overflow is indicated when the msb = 1. Manually setting the msb can cause an immediate interrupt. |

## 24.4 Functional Description

This section describes the use of some features of the performance monitor.

### 24.4.1 Performance Monitor Interrupt

PMCs can generate an interrupt on an overflow when the msb of a counter changes from 0 to 1. For the interrupt to be signalled, the condition enable bit (PMLCA*n*[CE]) and performance monitor interrupt enable bit (PMGC0[PMIE]) must be set. When an interrupt is signalled and the freeze-counters-on-enabled-condition-or-event bit (PMGC0[FCECE]) is set, PMGC0[FAC] is set by hardware and all of the registers are frozen. Software can clear the interrupt condition by resetting the performance monitor and clearing the most significant bit of the counter that generated the overflow.

## 24.4.2 Event Counting

Using the control registers described in Section 24.3.2, “Control Registers,” the twelve PMCs can count the occurrences of specific events. The 64-bit PMC0 is designated to count only clock cycles. However, to provide flexibility, a total of 64 reference events can be counted on any of the 32-bit PMCs (PMC1–PMC9). Additionally, up to 64 unique events can be counted on each 32-bit counter.

The performance monitor must be reset before event counting sequences. The performance monitor can be reset by first freezing one or more counters and then clearing the freeze condition to allow the counters to count according to the settings in the performance monitor registers.þCounters can be frozen individually by setting PMLCAn[FC] bits, or simultaneously by setting PMGC0[FAC]. Simply clearing these freeze bits will then allow the performance monitor to begin counting based on the register settings.þ

Note that using PMLCAn[FC] to reset the performance monitor resets only the specified counter. Performance monitor registers can be configured through reads or writes while the counters are frozen as long as freeze bits are not cleared by the register accesses.

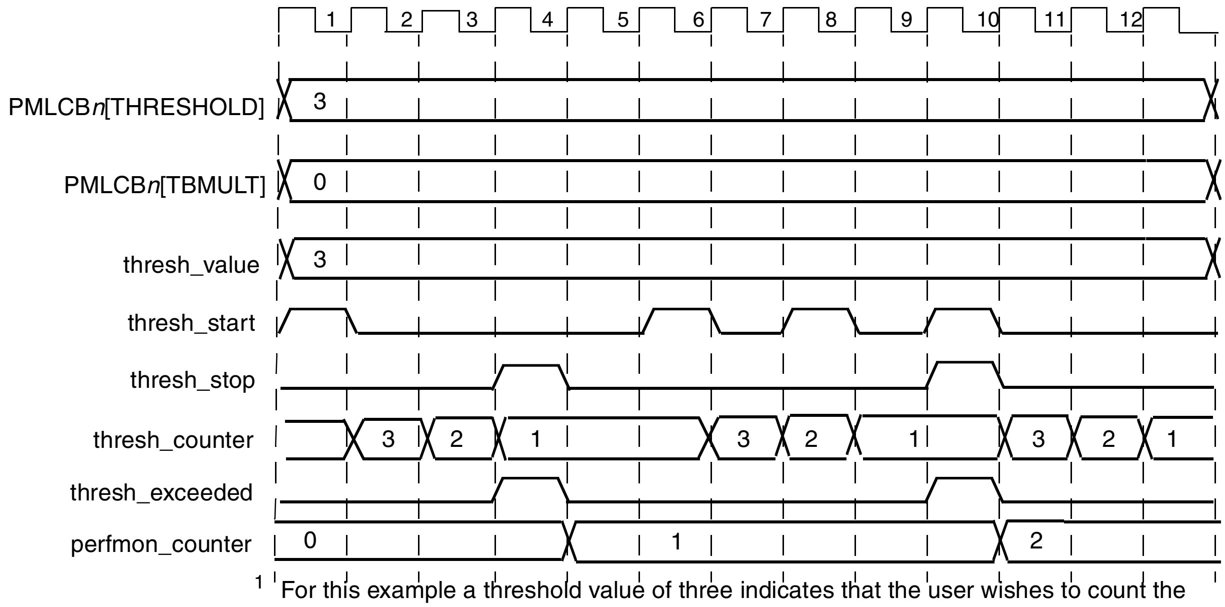
## 24.4.3 Threshold Events

The threshold feature allows characterization of events that can take a variable number of clock cycles to occur. Threshold events are counted only if the latency is greater than the threshold value specified in PMLCB*n*[THRESHOLD]. There are two types of threshold events.

The first type of threshold events are duration threshold events. For duration threshold event sequences, the PMC increments only when the duration of the event is equal to or greater than the threshold value. The threshold value is scaled by a multiple specified in PMLCB*n*[TBMULT].

A duration threshold event requires two signals: The first indicates when a threshold event sequence begins, and the second indicates when it ends. An internal counter determines when the threshold count is exceeded and when the PMC can increment. This internal counter decrements during a threshold event sequence until it reaches the value of one. A new sequence cannot begin until the current one completes. Additional threshold start signals are ignored during a sequence until a threshold stop signal occurs. If both a start and stop signal are asserted during the same cycle in a current sequence, the stop terminates the current sequence and the start signals the beginning of a new one. However, if both signals are asserted during the same cycle while not in a current event sequence, both signals are ignored. Figure 24-9 is a timing diagram for duration threshold event counting.

An illegal condition exists if the threshold value obtained from PMLCB*n*[THRESHOLD] and PMLCB*n*[TBMULT] is less than two. Under these conditions the intent of threshold counting is ambiguous.



[1] For this example a threshold value of three indicates that the user wishes to count the number of times a particular event lasts three cycles or longer.

**Figure 24-9. Duration Threshold Event Sequence Timing Diagram**

The second type of threshold event is the quantity threshold event. For these types of threshold event sequences the performance monitor counter is only incremented when the specified threshold event exceeds the threshold value. These events do not use the multiplier register field (PMLCB*n*[TBMULT]) like the duration threshold events. This type of threshold event is generally used to monitor the usage of buffers and queues. For example, the usage of a specific queue could be characterized by measuring the amount of time the queue is completely full or partially full. For this example the threshold field would be used to specify how many entries are required to be valid in the queue for that event to be counted.

## 24.4.4 Chaining

By configuring one counter to increment each time another counter overflows, several counters can be chained together to provide event counts larger than 32 bits. Each counter in a chain adds 32 bits to the maximum count. The register chaining sequence is not arbitrary and is specified indirectly by selecting the register overflow event to be counted. Selecting an event has the effect of selecting a source register because all available chaining events, as shown in Table 24-10, are dedicated to specific registers.

Note that the chaining overflow event occurs when the counter reaches its maximum value and wraps, not when the register's msb is set. For this overflow to occur, PMLCA*n*[CE] should be cleared to avoid signalling an interrupt when the counter's most significant bit is set. Note that several cycles may be required for the chained counters to reflect the true count because of the internal delay between when an overflow occurs and a counter increments.

## 24.4.5 Triggering

Triggering allows one counter to start or stop counting on the change of another counter or on the overflow of another counter. More specifically, if PMC1 is set to start or stop counting as a result of a change or overflow in counter PMC2, then counter PMC2 must be identified in the local control register of counter

PMC1. This is done by appropriately setting the trigger-on select bit or trigger-off select bit (PMLCB1[TRIGOFFSEL] or PMLCB1[TRIGONSEL]). Additionally, the condition that triggers the counter must be selected by configuring the corresponding control bits (PMLCB1[TRIGONCNTL] or PMLCB1[TRIGOFFCNTL]). Assuming the counter is enabled by other control register settings, the counter increments (or freezes) when its specified event occurs after the trigger-on (or off) condition occurs.

When trigger on and trigger off are both selected, the trigger-off condition is ignored until the trigger-on condition has occurred. Furthermore, when a trigger-off condition occurs, the counter state is preserved; it is not restarted by subsequent trigger-on conditions.

Triggering is disabled when the counter’s trigger-select bits specify itself as the trigger source. Similarly, triggering is disabled when the trigger control bits are cleared.

## 24.4.6 Burstiness Counting

The burstiness counting feature makes it easier to characterize events that occur in rapid succession followed by a relatively long pause. As shown in Table 24-9, event bursts are defined by size, granularity, and distance.

**Table 24-9. Burst Definition**

| Parameter | Description | Register Field |
|---|---|---|
| Size | The minimum number of events constituting a burst | PMLCA*n* [BSIZE] |
| Granularity | The maximum time between individual events counted as members of the same burst | PMLCA*n* [BGRAN] |
| Distance | The minimum time between bursts | PMLCA*n* [BDIST] x<br>PMLCB*n*[TBMULT] |

Figure 24-10 shows the relationships between size, granularity, and distance. Burstiness counting can be performed for all events except threshold events.

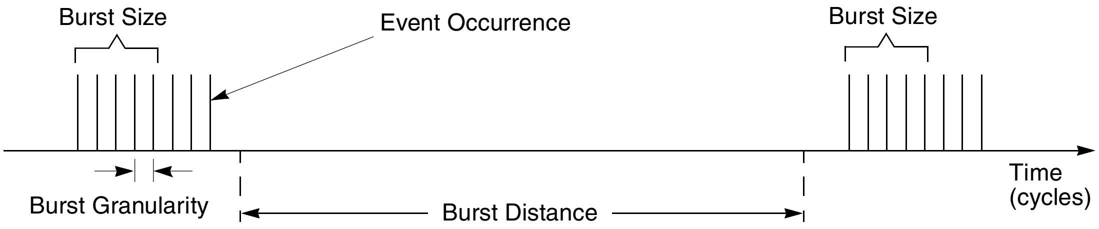


**Figure 24-10. Burst Size, Distance, Granularity, and Burstiness Counting**

The burstiness size field (PMLCA*n*[BSIZE]) specifies the minimum number of event occurrences that constitute a burst. A burst is identified when the number of event occurrences equals or exceeds PMLCA*n*[BSIZE]. Furthermore, these individual event occurrences must be separated by no more clock cycles than the value in the burstiness granularity field (PMLCA*n*[BGRAN]). Note that, although a burst is identified when the minimum number of events occurs, it is not counted until the burst sequence has ended. A burst sequence ends when the specified burstiness granularity is exceeded, at which point the last valid event has occurred for that sequence.

PMLCA*n*[BGRAN] specifies the maximum number of cycles between individual events for them to qualify as members of the same burst sequence.

The burstiness distance field (PMLCA*n*[BDIST]) and threshold/burstiness multiplier field (PMLCB*n*[TBMULT]) specify the acceptable number of cycles between the end of a burst sequence and the beginning of a new sequence for a group of event occurrences to be counted as an individual burst. The product of the burstiness distance field and the threshold/burstiness multiplier field determine the burstiness distance value used to determine when another burst sequence can begin. Note that the burst distance count begins when a new burst sequence ends and the PMC is incremented. No new burst sequence may begin until the burst distance count has reached zero. After the burst distance count reaches zero, it holds the zero value indicating that a new burst sequence can be counted. The burst distance count begins again when a new burst sequence is identified and counted.

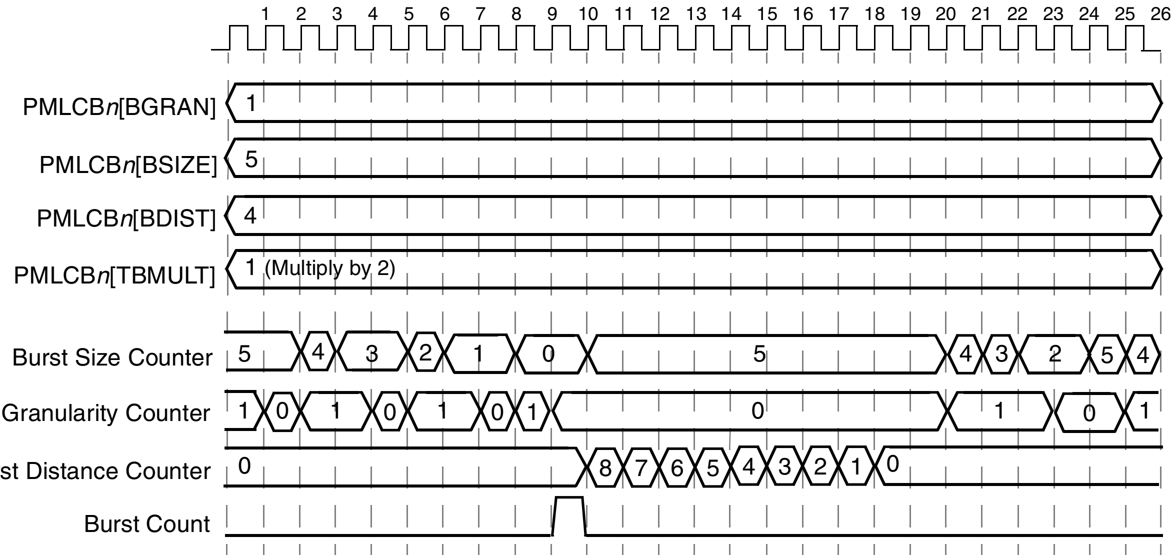
Burstiness counting is disabled when the definition of a burst is ambiguous, that is, when the burst size field is less than two, or the burst distance is zero. When burstiness counting is disabled, regular counting is allowed.

Figure 24-10 shows that the burst distance is measured from the end of one burst sequence and that a new burst sequence may not begin until the burst distance count expires.

Three internal counters track the different values required for burstiness counting.

- Burstiness size is monitored by a counter. It is loaded with the value specified in the local control register when the burst granularity counter and the burst distance counters reach zero, and no new event is occurring. It always decrements when the following conditions occur: its value is not already zero, an event occurs, and the burst distance count equals zero.
- Burstiness granularity is monitored by a counter that is loaded with the specified value in the local control register on the rising edge of an event occurrence whenever the burst distance count equals zero. The granularity counter is decremented (if it has not already reached zero) when an event is not occurring and burst distance count equals zero.
- Burstiness distance is measured by a counter that is loaded with the product of PMLCB*n*[BDIST] and PMLCB*n*[TBMULT] when a burst sequence has been identified and counted. This counter is decremented when burstiness counting is enabled (and the counter has not already reached zero).

A burst is counted at the end of a burst sequence when the three burst parameter counters are all equal to zero. Figure 24-11 shows a burstiness counting example.



[1] For this example: count bursts of 5 event occurrences, burst granularity of 1, and acceptable distance between bursts of 8 (product of TBMULT and BDIST).

**Figure 24-11. Burstiness Counting Timing Diagram**

## 24.4.7 Performance Monitor Events

Table 24-10 lists performance monitor events specified in PMLCA1–PMLC9.

The event assignment column indicates the event’s type and number, using the following formats:

- Ref:#—Reference events are shared across counters PMC1–PMC9. The number indicates the event. For example, Ref:6 means that PMC1–PMC9 share reference event 6.
- C[0–9]:#—Counter-specific events. C8 indicates an event assigned to PMC8. Thus C8:126 means PMC8 is assigned event 126 (PIC interrupt wait cycles).

Counter events not specified in Table 24-10 are reserved.

**NOTE**

Events generated by peripherals attached to the on-chip network (DMA, PCI, PCI Express) are counted twice, because these peripherals run at a slower clock rate than peripherals connected directly to the peripheral bus (for example, DDR or eTSEC).

### Table 24-10. Performance Monitor Events

| Event Counted | Number | Description of Event Counted |
|---|---|---|
| **General Events** | | |
| Nothing | Ref:0 | Register counter holds current value |
| System cycles | C:64 and Ref:20 | CCB (platform) clock cycles |
| **DDR Memory Controller Events** | | |
| Cycles a read is returning data from DRAM | Ref:19 | Each data beat returned to the memory controller on the DRAM interface |
| Cycles a write transfers data to DRAM | Ref:11 | Each data beat transferred to the DRAM |
| Pipelined read misses in the row open table | C1:121 | Row open table read misses issued while a read is outstanding |
| Pipelined read or write misses in the row open table | C2:64 | Row open table read or write misses issued while a read or write is outstanding |
| Non-pipelined read misses in the row open table | C3:124 | Row open table read misses issued when no reads are outstanding |
| Non-pipelined read or write misses in the row open table | C4:64 | Row open table read or write misses issued when no reads or writes are outstanding |
| Pipelined read hits in the row open table | C5:120 | Row open table read hits issued when a read is outstanding |
| Pipelined read or write hits in the row open table | C6:64 | Row open table read or write hits issued when a read or write is outstanding |
| Non-pipelined read hits in the row open table | C7:121 | Row open table read hits issued when no reads are outstanding |
| Non-pipelined read or write hits in the row open table | C8:64 | Row open table read or write hits issued when no reads or writes are outstanding |
| Forced page closings not caused by a refresh | C1:64 | Precharges issued to the DRAM for any reason except refresh. The possibilities are as follows:<br>• A new transaction must be issued to an already active bank and sub-bank that has a different row open.<br>• A new transaction must be issued, but the row open table is full and there is no bank/sub-bank match between the current transaction and the row open table.<br>• The BSTOPRE interval expired for an open row. |
| Row open table misses | C2:65 | Transactions that miss in the row open table |
| Row open table hits | C3:64 | Transaction that hit in the row open table |
| Force page closings | C4:65 | Forced page closings including those due to refreshes |
| Read-modify-write transactions due to ECC | C5:64 | If ECC is enabled and a transaction requires byte enables, a read-modify-write sequence is issued on the DRAM interface. |
| Forced page closings due to collision with bank and sub-bank | Ref:12 | Increments if a new transaction must be issued to an active bank and sub-bank that has a different row open |
| Reads or writes from core (data and inst) | Ref:13 | — |

Table 24-10. Performance Monitor Events (continued)

| Event Counted | Number | Description of Event Counted |
|---|---|---|
| Reads or writes from eTSEC 1 or eTSEC3 | C3:65 | — |
| Reads or writes from high speed interfaces (PCI and PEX2) | C3:66 | — |
| Reads or writes from high speed interfaces (PCI and PEX1–3) | C4:67 | — |
| Reads or writes from DMA | C5:66 | — |
| Reads or writes from Security | C6:69 | — |
| Row open table hits for reads or writes from core (data and inst) | Ref:14 | — |
| Row open table hits for reads or writes from eTSEC 1 or eTSEC3 | C6:65 | — |
| Row open table hits for reads or writes from high speed interfaces (PCI and PEX2) | C6:66 | — |
| Row open table hits for reads or writes from high speed interfaces (PCI and PEX1–3) | C7:65 | — |
| Row open table hits for reads or writes from DMA | C8:66 | — |
| Row open table hits for reads or writes from Security | C7:68 | — |
| **DMA Controller Events** | | |
| Channel 0 read request | C1:66 | DMA channel 0 read request active in the system |
| Channel 1 read request | C2:69 | DMA channel 1 read request active in the system |
| Channel 2 read request | C3:68 | DMA channel 2 read request active in the system |
| Channel 3 read request | C4:70 | DMA channel 3 read request active in the system |
| Channel 0 write request | C1:67 | DMA channel 0 write request active in the system |
| Channel 1 write request | C2:70 | DMA channel 1 write request active in the system |
| Channel 2 write request | C3:69 | DMA channel 2 write request active in the system |
| Channel 3 write request | C4:71 | DMA channel 3 write request active in the system |
| Channel 0 descriptor request | C5:105 | DMA channel 0 descriptor request active in the system |
| Channel 1 descriptor request | C6:108 | DMA channel 1 descriptor request active in the system |
| Channel 2 descriptor request | C7:105 | DMA channel 2 descriptor request active in the system |
| Channel 3 descriptor request | C8:105 | DMA channel 3 descriptor request active in the system |
| Channel 0 read DW or less | C1:68 and C5:117 | DMA channel 0 read double word valid |
| Channel 1 read DW or less | C2:71 and C6:122 | DMA channel 1 read double word valid |

Table 24-10. Performance Monitor Events (continued)

| Event Counted | Number | Description of Event Counted |
|---|---|---|
| Channel 2 read DW or less | C3:70 and C7:118 | DMA channel 2 read double word valid |
| Channel 3 read DW or less | C4:72 and C8:116 | DMA channel 3 read double word valid |
| Channel 0 write DW or less | C1:69 | DMA channel 0 write double word valid |
| Channel 1 write DW or less | C2:72 | DMA channel 1 write double word valid |
| Channel 2 write DW or less | C3:71 | DMA channel 2 write double word valid |
| Channel 3 write DW or less | C4:73 | DMA channel 3 write double word valid |
| **e500 Coherency Module (ECM) Events** | | |
| ECM request wait core | C8:77 | Asserted for every cycle core request occurs |
| ECM request wait I$^2$C/Security | C7:77 | Asserted for every cycle I$^2$C/Security request occurs |
| ECM request wait PEX1–3/DMA | C5:80 | Asserted for every cycle PEX1-3/DMA request occurs |
| ECM request wait eTSEC1/3 | C6:80 | Asserted for every cycle eTSEC1/3 request occurs |
| ECM request wait USB1–3/eSDHC/SATA1–2 | C4:84 | Asserted for every cycle USB1–3/eSDHC/SATA1–2 request occurs |
| ECM dispatch | Ref:15 | ECM dispatch (includes address only's)<br>**Note:** All ECM dispatch events are for committed dispatches |
| ECM dispatch from USB2 | C3:83 | — |
| ECM dispatch from eTSEC1 | C4:85 | — |
| ECM dispatch from eTSEC3 | C9:80 | — |
| ECM dispatch from SATA1 | C8:99 | — |
| ECM dispatch from SATA2 | C6:81 | — |
| ECM dispatch from eSDHC | C7:80 | — |
| ECM dispatch from PCI | C8:81 | — |
| ECM dispatch from PEX2 | C7:94 | — |
| ECM dispatch from PEX1 | C7:78 | — |
| ECM dispatch from PEX3 | C9:81 | — |
| ECM dispatch from DMA | C9:82 | — |
| ECM dispatch from Security | C5:83 | — |
| ECM dispatch from USB1 | C5:82 | — |
| ECM dispatch from USB3 or boot sequencer | C6:82 | — |
| ECM dispatch to DDR | C7:79 | ECM dispatch to DDR (excludes address only) |
| ECM dispatch to L2 | C3:80 | — |
| ECM dispatch to SRAM | C8:79 | — |
| ECM dispatch to eLBC | C9:83 | — |

**Table 24-10. Performance Monitor Events (continued)**

| Event Counted | Number | Description of Event Counted |
|---|---|---|
| ECM dispatch to PCI | C2:84 | — |
| ECM dispatch to PEX2 | C1:81 | — |
| ECM dispatch to PEX1 | C2:85 | — |
| ECM dispatch to PEX3 | C4:87 | — |
| ECM dispatch to CCSR | C9:87 | — |
| ECM dispatch write | C7:81 | — |
| ECM dispatch write allocate | C1:82 | — |
| ECM dispatch read | C2:86 | — |
| ECM dispatch read atomic | C3:86 | — |
| ECM data bus grant DDR1 | C1:83 | Number of data bus grants from the ECM global data bus arbiter |
| ECM data bus grant I$^2$C/Security | C3:87 | Number of data bus grants from the ECM global data bus arbiter |
| ECM data bus grant PEX1–3/DMA | C4:89 | Number of data bus grants from the ECM global data bus arbiter |
| ECM data bus grant eLBC | Ref:16 | Number of data bus grants from the ECM global data bus arbiter |
| ECM data bus grant eTSEC1 and eTSEC3 | Ref:17 | Number of data bus grants from the ECM global data bus arbiter |
| ECM data bus grant TargetQ | C2:88 | Number of data bus grants from the ECM global data bus arbiter |
| ECM data bus grant USB1–3/eSDHC/SATA1–2 | Ref:18 | Number of data bus grants from the ECM global data bus arbiter |
| ECM global data bus beat | C5:86 | — |
| **Security/Boot Sequencer Events** | | |
| Security/boot sequencer requests | C1:119 | Number of security/sequencer requests (total) |
| Security/boot sequencer read requests | C2:76 | Number of security/sequencer read requests |
| Security/boot sequencer data beats | C3:75 | Number of security/sequencer data beats (total) |
| Security/boot sequencer read data beats | C4:79 | Number of security/sequencer read data beats |
| Security/boot sequencer less than 32 bytes | C7:76 | Number of security/sequencer requests less than 32 bytes |
| **USB1–3/eSDHC/SATA1–2 Events** | | |
| USB1–3/eSDHC/SATA1–2 requests | C1:123 | Number of USB1–3/eSDHC/SATA1–2 requests (total) |
| USB1–3/eSDHC/SATA1–2 read requests | C2:77 | Number of USB1–3/eSDHC/SATA1–2 read requests |
| USB1–3/eSDHC/SATA1–2 data beats | C3:77 | Number of USB1–3/eSDHC/SATA1–2 data beats (total) |
| USB1–3/eSDHC/SATA1–2 read data beats | C4:80 | Number of USB1–3/eSDHC/SATA1–2 read data beats |
| USB1–3/eSDHC/SATA1–2 request less than 32 bytes | C7:75 | Number of USB1–3/eSDHC/SATA1–2 requests less than 32 bytes |

### Table 24-10. Performance Monitor Events (continued)

| Event Counted | Number | Description of Event Counted |
|---|---|---|
| **Interrupt Controller (PIC) Events** | | |
| PIC total interrupt count | Ref:26 | Total number of interrupts serviced |
| PIC interrupt wait cycles | C8:126 | Counts cycles when an interrupt waits to be acknowledge |
| PIC interrupt service cycles | C2:83 | Number of cycles there is an interrupt currently being serviced. |
| PIC interrupt select 0 (duration threshold) | C1:120 | THRESHOLD: select 0–3: interrupt count over threshold. (Note: only unmasked, nonzero priority requests are acknowledged). The four interrupts are selected through register pairs, PM0MR*n*–PM3MR*n*. See Section 9.3.4, “Performance Monitor Mask Registers (PMMRs).” |
| PIC interrupt select 1 (duration threshold) | C3:123 | |
| PIC interrupt select 2 (duration threshold) | C5:119 | |
| PIC interrupt select 3 (duration threshold) | C6:124 | |
| **PCI Events** | | |
| PCI clock cycles | Ref:28 | — |
| PCI inbound memory reads | C1:126 | Includes all read types. |
| PCI inbound memory writes | C2:101 | — |
| PCI inbound config reads | C3:127 | — |
| PCI inbound config writes | C4:101 | — |
| PCI outbound memory reads | C5:94 | Includes all read types. |
| PCI outbound memory writes | C6:96 | Number of PCI outbound memory writes |
| PCI outbound I/O reads | C3:101 | — |
| PCI outbound I/O writes | C4:102 | — |
| PCI outbound config reads | C7:90 | Number of PCI outbound config reads |
| PCI outbound config writes | C8:90 | — |
| PCI inbound 32-bit read data beats | C1:94 | — |
| PCI inbound 32-bit write data beats | C2:102 | — |
| PCI outbound 32-bit read data beats | C3:102 | — |
| PCI outbound 32-bit write data beats | C4:103 | — |
| PCI total transactions | C7:93 | Includes 32- and 64-bit transactions. |
| PCI inbound purgeable reads | C2:66 | — |
| PCI inbound (speculative reads) purgeable reads discarded | C8:127 | — |
| PCI idle cycles | C1:95 | — |
| PCI dual address cycles | C2:104 | — |
| PCI internal cycles | C3:103 | — |

Table 24-10. Performance Monitor Events (continued)

| Event Counted | Number | Description of Event Counted |
|---|---|---|
| PCI inbound memory read | C1:98 | — |
| PCI inbound memory readline | C2:108 | — |
| PCI inbound memory read multiple | C3:106 | — |
| PCI outbound memory reads | C4:107 | Number of PCI outbound memory reads |
| PCI outbound memory read lines | C5:100 | Number of PCI outbound memory read lines |
| PCI wait | C1:99 | $\overline{\text{PCI\_IRDY}}$, $\overline{\text{PCI\_TRDY}}$ not both asserted |
| PCI snoopable | C1:96 | — |
| PCI write stash | C2:106 | — |
| PCI write stash with lock | C3:105 | — |
| PCI read unlock | C4:106 | — |
| PCI byte enable transactions | C1:97 | — |
| PCI non-byte enable transactions | C2:107 | — |
| **eTSEC 1 Events** | | |
| DMA write data beats | C3:109 | DMA write data beats |
| DMA read data beats | C4:110 | DMA read data beats |
| DMA Write Request | C5:106 | DMA Write Request |
| DMA Read Request | C6:109 | DMA Read Request |
| Number of dropped frames | C9:88 | Number of dropped frames |
| TxBD read lifetime (Duration Threshold) | Ref:34 | TxBD read lifetime |
| RxBD read lifetime (Duration Threshold) | Ref:38 | RxBD read lifetime |
| TxBD write lifetime (Duration Threshold) | Ref:42 | TxBD write lifetime |
| RxBD write lifetime (Duration Threshold) | Ref:46 | RxBD write lifetime |
| Read data lifetime (Duration Threshold) | Ref:50 | Read data lifetime |
| Rx IP packets checked for checksum | C9:92 | Rx IP packets checked for checksum |
| TX IP packet with checksum | C1:105 | TX IP packet with checksum |
| TX TCP/UDP packet with checksum | C2:113 | TX TCP/UDP packet with checksum |
| TCP/UDP packets checked for c.s. | C3:114 | TCP/UDP packets checked for c.s. |
| IP or TCP/UDP Rx checksum error | C4:115 | IP or TCP/UDP Rx checksum error |
| Number of rejected frames by filer | C5:111 | Number of rejected frames by filer |
| Number of rejected frames due to filer error | C6:114 | Number of rejected frames due to filer error |
| Number of cycles Rx FIFO > 1/4 full | C5:110 | Number of cycles Rx FIFO > 1/4 full |
| Number of cycles Rx FIFO > 1/2 full | C6:113 | Number of cycles Rx FIFO > 1/2 full |

**Table 24-10. Performance Monitor Events (continued)**

| Event Counted | Number | Description of Event Counted |
|---|---|---|
| Number of cycles Rx FIFO > 3/4 full | C7:110 | Number of cycles Rx FIFO > 3/4 full |
| Number of cycles Rx FIFO = full | C8:110 | Number of cycles Rx FIFO = full |
| Number of accepted frames matc | C9:89 | — |
| Number of accepted frams to station address | C8:101 | — |
| Number of accepted unicaset frames via has | C7:96 | — |
| Number of accepted group frames via hash | C6:89 | — |
| Number of accepted frams via exact match | C5:85 | — |
| Number of rejected frames at layer 2 | C4:105 | — |
| Number of RX interrupts signalled | C3:82 | — |
| Number of TX interrupts signalled | C2:110 | — |
| RX data write lifetime (Duration Threshold) | Ref:43 | — |
| Number of RX packets received while RX FIFO is full | Ref:47 | — |
| **eTSEC 3 Events** | | |
| DMA write data beats | C7:108 | DMA write data beats |
| DMA read data beats | C8:108 | DMA read data beats |
| DMA Write Request | C9:90 | DMA Write Request |
| DMA Read Request | C1:103 | DMA Read Request |
| Number of dropped frames | C4:112 | Number of dropped frames |
| TxBD read lifetime (Duration Threshold) | Ref:36 | TxBD read lifetime |
| RxBD read lifetime (Duration Threshold) | Ref:40 | RxBD read lifetime |
| TxBD write lifetime (Duration Threshold) | Ref:44 | TxBD write lifetime |
| RxBD write lifetime (Duration Threshold) | Ref:48 | RxBD write lifetime |
| Read data lifetime (Duration Threshold) | Ref:52 | Read data lifetime |
| Rx IP packets checked for checksum | C6:116 | Rx IP packets checked for checksum |
| TX IP packet with checksum | C7:112 | TX IP packet with checksum |
| TX TCP/UDP packet with checksum | C8:112 | TX TCP/UDP packet with checksum |
| TCP/UDP packets checked for c.s. | C9:94 | TCP/UDP packets checked for c.s. |
| IP or TCP/UDP Rx checksum error | C1:106 | IP or TCP/UDP Rx checksum error |
| Number of rejected frames by filer | C2:114 | Number of rejected frames by filer |

Table 24-10. Performance Monitor Events (continued)

| Event Counted | Number | Description of Event Counted |
|---|---|---|
| Number of rejected frames due to filer error | C3:116 | Number of rejected frames due to filer error |
| Number of cycles Rx FIFO > 1/4 full | C9:97 | Number of cycles Rx FIFO > 1/4 full |
| Number of cycles Rx FIFO > 1/2 full | C1:109 | Number of cycles Rx FIFO > 1/2 full |
| Number of cycles Rx FIFO > 3/4 full | C2:116 | Number of cycles Rx FIFO > 3/4 full |
| Number of cycles Rx FIFO = full | C3:118 | Number of cycles Rx FIFO = full |
| Number of accepted frames matc | C9:91 | — |
| Number of accepted frams to station address | C8:102 | — |
| Number of accepted unicaset frames via has | C7:100 | — |
| Number of accepted group frames via hash | C6:90 | — |
| Number of accepted frams via exact match | C5:95 | — |
| Number of rejected frames at layer 2 | C4:111 | — |
| Number of RX interrupts signalled | C3:85 | — |
| Number of TX interrupts signalled | C2:112 | — |
| RX data write lifetime (Duration Threshold) | Ref:45 | — |
| Number of RX packets received while RX FIFO is full | Ref:49 | — |
| **PCI Express 1 Events** | | |
| Inbound G2PI read | C8:119 | A single pulse to indicate an inbound PCI Express 1 read has occurred. |
| Inbound G2PI write | C9:101 | A single pulse to indicate an inbound PCI Express 1 write has occurred. |
| Inbound G2PI data | C5:124 | A level signal to indicate the amount of data transferred if any for inbound PCI Express 1 request. Active for every beat of PCI Express 1 data. |
| Outbound G2PI read | C6:126 | A single pulse to indicate an outbound PCI Express 1 read has occurred. |
| Outbound G2PI write | C7:125 | A single pulse to indicate an outbound PCI Express 1 write has occurred. |
| Outbound G2PI data | C8:124 | A level signal to indicate the amount of data transferred if any for outbound PCI Express 1 request. Active for every beat of PCI Express 1 data. |

Table 24-10. Performance Monitor Events (continued)

| Event Counted | Number | Description of Event Counted |
|---|---|---|
| Inbound Static Queue 0 start (Duration Threshold) | Ref:54 | Lifetime of ISQ entry 0 or 6. |
| Outbound Static Queue 0 start (Duration Threshold) | Ref:55 | Lifetime of OSQ entry 0. |
| **PCI Express 2 Events** | | |
| Inbound G2PI read | C8:70 | A single pulse to indicate an inbound PCI Express 2 read has occurred. |
| Inbound G2PI write | C9:64 | A single pulse to indicate an inbound PCI Express 2 write has occurred. |
| Inbound G2PI data | C5:74 | A level signal to indicate the amount of data transferred if any for inbound PCI Express 2 request. Active for every beat of PCI Express 2 data. |
| Outbound G2PI read | C6:74 | A single pulse to indicate an outbound PCI Express 2 read has occurred. |
| Outbound G2PI write | C7:73 | A single pulse to indicate an outbound PCI Express 2 write has occurred. |
| Outbound G2PI data | C8:72 | A level signal to indicate the amount of data transferred if any for outbound PCI Express 2 request. Active for every beat of PCI Express 2 data. |
| Inbound Static Queue 0 start (Duration Threshold) | Ref:56 | Lifetime of ISQ entry 0 or 6. |
| Outbound Static Queue 0 start (Duration Threshold) | Ref:57 | Lifetime of OSQ entry 0. |
| **PCI Express 3 Events** | | |
| Inbound G2PI read | C8:71 | A single pulse to indicate an inbound PCI Express 3 read has occurred. |
| Inbound G2PI write | C9:65 | A single pulse to indicate an inbound PCI Express 3 write has occurred. |
| Inbound G2PI data | C5:75 | A level signal to indicate the amount of data transferred if any for inbound PCI Express 3 request. Active for every beat of PCI Express 3 data. |
| Outbound G2PI read | C6:75 | A single pulse to indicate an outbound PCI Express 3 read has occurred. |
| Outbound G2PI write | C7:83 | A single pulse to indicate an outbound PCI Express 3 write has occurred. |
| Outbound G2PI data | C8:73 | A level signal to indicate the amount of data transferred if any for outbound PCI Express 3 request. Active for every beat of PCI Express 3 data. |
| Inbound Static Queue 0 start (Duration Threshold) | Ref:58 | Lifetime of ISQ entry 0 or 6. |

**Table 24-10. Performance Monitor Events (continued)**

| Event Counted | Number | Description of Event Counted |
|---|---|---|
| Outbound Static Queue 0 start (Duration Threshold) | Ref:59 | Lifetime of OSQ entry 0. |
| **Local Bus Events** | | |
| Atomic reservation time-outs for ECM port | C6:118 | — |
| Cycles a read is taking in GPCM | C1:117 | — |
| Cycles a read is taking in UPM | C2:122 | — |
| Cycles a write is taking in GPCM | C4:120 | — |
| Cycles a write is taking in UPM | C5:114 | — |
| **L2 Cache/SRAM Events** | | |
| Core instruction accesses to L2 that hit | C2:123 | — |
| Core instruction accesses to L2 that miss | Ref:23 | — |
| Core data accesses to L2 that hit | C4:121 | — |
| Core data accesses to L2 that miss | C5:115 | — |
| Non-core burst write to L2 (cache external write or SRAM) | C6:120 | — |
| Non-core non-burst write to L2 | C7:116 | — |
| Noncore write misses cache external write window and SRAM memory range | Ref:24 | — |
| Non-core read hit in L2 | C1:118 | — |
| Non-core read miss in L2 | Ref:25 | — |
| L2 allocations based on core-initiated accesses. The data may come from any source. | C2:124 | — |
| L2 retries due to full write queue | C3:122 | — |
| L2 retries due to address collision | C4:122 | — |
| L2 failed lock attempts due to full set | C5:116 | — |
| L2 victimizations of valid lines | C6:121 | — |
| L2 invalidations of lines | C7:117 | — |
| L2 clearing of locks | Ref:22 | — |
| **Debug Events** | | |
| External event | C3:125 | Number of cycles trig_in pin is asserted |
| Watchpoint monitor hits | C2:125 | — |
| Trace buffer hits | C1:122 | — |
| **DUART Events** | | |

**Table 24-10. Performance Monitor Events (continued)**

| Event Counted | Number | Description of Event Counted |
|---|---|---|
| UART0 baud rate | C1:127 | — |
| UART1 baud rate | C5:127 | — |
| **Chaining Events** | | |
| PMC0 carry-out | Ref:1 | PMC0[0] 1-to-0 transitions. |
| PMC1 carry-out | Ref:2 | PMC1[0] 1-to-0 transitions. Reserved for PMC1. |
| PMC2 carry-out | Ref:3 | PMC2[0] 1-to-0 transitions. Reserved for PMC2. |
| PMC3 carry-out | Ref:4 | PMC3[0] 1-to-0 transitions. Reserved for PMC3. |
| PMC4 carry-out | Ref:5 | PMC4[0] 1-to-0 transitions. Reserved for PMC4. |
| PMC5 carry-out | Ref:6 | PMC5[0] 1-to-0 transitions. Reserved for PMC5. |
| PMC6 carry-out | Ref:7 | PMC6[0] 1-to-0 transitions. Reserved for PMC6. |
| PMC7 carry-out | Ref:8 | PMC7[0] 1-to-0 transitions. Reserved for PMC7. |
| PMC8 carry-out | Ref:9 | PMC8[0] 1-to-0 transitions. Reserved for PMC8. |
| PMC9 carry-out | Ref:10 | PMC9[0] 1-to-0 transitions. Reserved for PMC9. |

## 24.4.8 Performance Monitor Examples

Table 24-12 contains sample register settings for the four supported modes.

- Simple event performance monitoring example
- Triggering event performance monitoring example
- Threshold event performance monitoring example
- Burstiness event performance monitoring example

The settings in Table 24-11 are identical for all four examples.

**Table 24-11. PMGC0 and PMLCA*n* Settings**

| Field | Setting | Reason |
|---|---|---|
| PMGC0[FAC] | 0 | Counters must not be frozen. |
| PMGC0[PMIE] | 1 | Performance monitor interrupts are enabled |
| PMGC0[FCECE] | 1 | Counters should be frozen when an interrupt is signalled. |
| PMLCA*n*[FC] | 0 | Counters cannot be frozen for counting. |
| PMLCA*n*[CE] | 1 | Overflow condition enable is required to allow interrupt signalling. |

For simple event counting, a non-threshold event is selected in PMLCA*n*[EVENT] and all other features are disabled by clearing all register fields except for CE.

For the triggering example any event can be selected in PMLCA*n*[EVENT]. All other features are disabled by clearing these register fields except for CE to allow interrupt signalling. If PMLCB*n*[TRIGONSEL] is

3 and PMLCB*n*[TRIGOFFSEL] is 5, the counter begins and ends counting based on the conditions in counters three and five. Furthermore, if PMLCB*n*[TRIGONCNTL] is 1, the counter begins counting when PMC3 changes value. According to the setting in PMLCB*n*[TRIGOFFCNTL], the counter ends counting when PMC5 overflows. Also, although the register settings for PMC5 is not shown, PMLCA*n*[CE] for this counter must be cleared so that interrupt signalling is not enabled and the counter does not freeze when it overflows.

For threshold counting, a threshold event must be specified in PMLCA*n*[EVENT]. For this example, the duration threshold value is scaled by two because PMLCB*n*[TBMULT] is one. All other features are disabled by clearing the appropriate fields.

Any non-threshold event can use the burstiness feature. For burstiness counting, values for PMLCA*n*[BSIZE,BGRAN,BDIST] and PMLCB*n*[TBMULT] must be specified.

**Table 24-12. Register Settings for Counting Examples**

| Register | Register Field | Simple Event | Triggering | Threshold | Burstiness |
|---|---|---|---|---|---|
| PMGC0 | FAC | 0 | 0 | 0 | 0 |
| | PMIE | 1 | 1 | 1 | 1 |
| | FCECE | 1 | 1 | 1 | 1 |
| PMLCA*n* | FC | 0 | 0 | 0 | 0 |
| | CE | 1 | 1 | 1 | 1 |
| | EVENT | 89 | 68 | 39 | 2 |
| | BSIZE | 0 | 0 | 0 | 5 |
| | BGRAN | 0 | 0 | 0 | 1 |
| | BDIST | 0 | 0 | 0 | 8 |
| PMLCB*n* | TRIGONSEL | 0 | 3 | 0 | 0 |
| | TRIGOFFSEL | 0 | 5 | 0 | 0 |
| | TRIGONCNTL | 0 | 1 | 0 | 0 |
| | TRIGOFFCNTL | 0 | 2 | 0 | 0 |
| | TBMULT | 0 | 0 | 0 | 0 |
| | THRESHOLD | 0 | 0 | 3 | 0 |

The performance monitor must be reset before event counting sequences. The performance monitor can be reset by first freezing one or more counters and then clearing the freeze condition to allow the counters to count according to the settings in the performance monitor registers. Counters can be frozen individually by setting PMLCAn[FC] bits, or simultaneously by setting PMGC0[FAC]. Simply clearing these freeze bits will then allow the performance monitor to begin counting based on the register settings.þ

Note that using PMLCAn[FC] to reset the performance monitor resets only the specified counter. Performance monitor registers can be configured through reads or writes while the counters are frozen as long as freeze bits are not cleared by the register accesses.

# Chapter 25
# Debug Features and Watchpoint Facility

This chapter describes all customer-visible debug modes of the MPC8536E integrated device. The debug features on the MPC8536E pertain to these interfaces: the local bus controller (LBC), and the DDR SDRAM interface. In addition to the external interfaces, the MPC8536E provides triggering capabilities based on user-programmable events. The watchpoint and trace buffer also provide some visibility to internal buses. This chapter also describes context ID registers, useful for software debug, and describes the JTAG access port signals that comply with the IEEE 1149.1 boundary-scan specification.

## 25.1 Introduction

As shown in the block diagram of Figure 25-1, the MPC8536E device provides the following debug features (listed with references to sections of this chapter that describe them):

- DDR SDRAM interface debug (Section 25.4.2, "DDR SDRAM Interface Debug")
- Local bus controller (LBC) debug (Section 25.4.3, "Local Bus Interface Debug")
- Watchpoint monitor and trace buffer debug (Section 25.4.4, "Watchpoint Monitor," and Section 25.4.5, "Trace Buffer")

### 25.1.1 Overview

As shown in Figure 25-1, debug information is provided through the following interfaces: LBC, and DDR SDRAM. Limited visibility, through a 256 x 64 trace buffer, is also provided for the processor core interface. This visibility into internal device operation is useful for debugging application software through inverse assembly and reconstruction of the fetch stream.

The combination of a source ID (MSRCID[0:4]) and a data-valid signal (MDVAL) indicates that meaningful debug information is visible on either the local bus or DDR SDRAM interfaces. A logic analyzer can be programmed to capture data based on the values of MSRCID[0:4] and MDVAL.

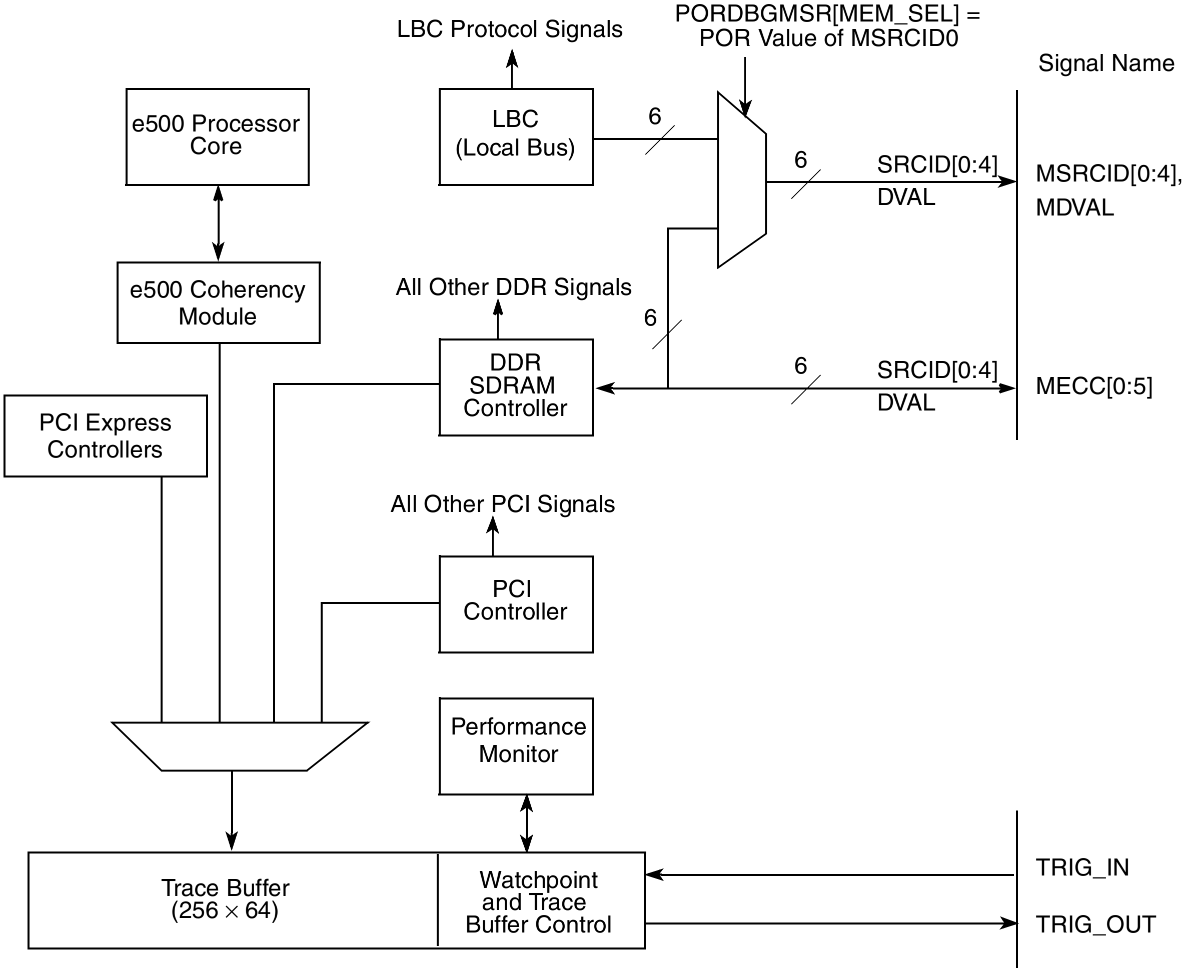


**Figure 25-1. Debug and Watchpoint Monitor Block Diagram**

Other system debugging is supported by the programmable triggering of the watchpoint monitor and trace buffer. Both can be triggered from one of the following three sources:

- Each other
- A performance monitor event
- An external source (through TRIG_IN).

The watchpoint monitor can be configured to assert TRIG_OUT when a programmed event occurs. The two context ID registers, described in Section 25.3.3, “Context ID Registers,” are useful for software debug.

### 25.1.2 Features

The principal features of the debug modes and the watchpoint monitor are as follows:

- LBC and DDR interface source ID and data-valid indicators
  - LBC or DDR SDRAM source ID can be selected to be driven onto MSRCID[0:4]
  - Source ID and data-valid indicators can be selected to be driven onto the error correcting code (ECC) pins of the DDR interface
- Watchpoint monitor that supports
  - Two-level triggering
  - Programmable external trigger (TRIG_OUT)
  - Interlocked with performance monitor to use its large number of counters
- Trace buffer features that support
  - Two-level triggering
  - Programmable external trigger (TRIG_OUT)
  - Interlocked with performance monitor to use its large number of counters
  - 256-entry trace buffer, 64 bits each
  - Programmable trace start and stop
  - Can function as a second watchpoint monitor
- Context ID registers that can be programmed to trigger events

### 25.1.3 Modes of Operation

The LBC, and DDR SDRAM interfaces all have debug modes, which are controlled by values on configuration inputs during the power-on reset (POR) sequence, as shown in Table 25-1.The DDR controller can also drive debug information on either MSRCID[0:4] or MECC[0:5]. See Section 25.4.1, "Source and Target ID," for additional information about the source ID information driven on the debug signals in these modes.

Note that both the watchpoint monitor and trace buffer also operate in a variety of modes.

**Table 25-1. POR Configuration Settings and Debug Modes**

| Configuration Signal | POR Value | Effect | Reference |
|---|---|---|---|
| MSRCID0 | 0 | Local bus SDRAM information appears on MSRCID[0:4] and MDVAL. | 25.1.3.1/25-4 |
| | 1 | Default value (internal pull-up resistor). DDR SDRAM information appears on MSRCID[0:4] and MDVAL. | |
| MSRCID1 | 0 | MECC[0:4] operate in debug mode and provide memory debug source ID and MECC5 provides data-valid information. | 25.1.3.2/25-4 |
| | 1 | Default value (internal pull-up resistor). MECC[0:4] operate in normal mode and provide DDR SDRAM error correcting code information. | |

### 25.1.3.1 Local Bus (LBC) Debug Mode

The LBC and the DDR SDRAM controller can drive debug information (source ID and data-valid indicator) onto MSRCID[0:4] and MDVAL. As shown in Table 25-1, the MSRCID0 value during POR controls multiplexing. If MSRCID0 is low when sampled during POR, the local bus SDRAM information appears on MSRCID[0:4] and MDVAL; otherwise, the DDR SDRAM debug information is presented.

### 25.1.3.2 DDR SDRAM Interface Debug Modes

MSRCID1 is sampled during POR to multiplex either ECC or debug information on the ECC pins of the DDR SDRAM interface. As shown in Table 25-1, if MSRCID1 is low during POR, the ECC pins operate in debug mode and provide memory debug source ID and data-valid information. MSRCID1 must be pulled low during POR to use the ECC pins in debug mode. If MSRCID1 is unconnected, an internal pull-up resistor ensures the ECC pins always source DDR SDRAM error correcting code information as their default power-on reset configuration.

**NOTE**

> If the DDR ECC pins are in debug mode (configured for debug during POR), ECC checking is disabled in the memory controller. In this case, MECC[0:4] do not provide ECC information and must not be connected to SDRAM devices.

### 25.1.3.3 Watchpoint Monitor Modes

The watchpoint monitor supports the following operating modes:

- Immediate trigger arming (one-level triggering)—The watchpoint monitor triggers as soon as the first trigger event occurs.
- Wait for trigger arming (two-level triggering)—The watchpoint monitor waits for a specific event before enabling (arming) the trigger logic. The monitor does not respond to trigger events until after the arming event occurs. This function is similar to two-level triggering on a logic analyzer.
- Assert TRIG_OUT on hit—The debug block can be programmed to assert the TRIG_OUT signal when a programmed watchpoint monitor event occurs. This signal can be used to trigger a logic analyzer.

### 25.1.3.4 Trace Buffer Modes

The trace buffer supports the following operating modes:

- Immediate trigger arming (one-level triggering)—The trace buffer triggers as soon as the first trigger event occurs.
- Wait for trigger arming (two-level triggering)—The trace buffer waits for a specific event before enabling (arming) the trigger logic. The trace buffer does not respond to trigger events until after the arming event occurs. This function is similar to two-level triggering on a logic analyzer.
- Specific interface selection—The trace buffer can be programmed to trace one of several internal interfaces.

- Specific event selection—The trace buffer can be programmed to trace on the occurrence of one or several concurrent events.
- Specific trace selection—To facilitate trace data filtering, the trace buffer can be configured to capture data under the following conditions:
  - On every cycle in which a valid transaction is present on the selected interface
  - Only when the programmed trace event is detected
- Programmable trace stop—The trace buffer may be programmed to stop tracing when a programmed stop-tracing event occurs or when the 256-entry buffer is full.

## 25.2 External Signal Description

This section provides information about all the external signals associated with the various MPC8536E debug functions.

As shown in Table 25-1, the MPC8536E has several signals that are sampled during POR to determine the configuration of the phase-locked loop clock mode and the ROM, flash, and dynamic memory. See Chapter 4, “Reset, Clocking, and Initialization.”

To facilitate system testing, the MPC8536E provides a JTAG test access port (TAP) that complies with the IEEE 1149.1 boundary-scan specification. This section also describes JTAG TAP signals.

### 25.2.1 Overview

All the signals associated with device debug features are summarized in Table 25-2, listed with a reference to the page number of the section with more information. The detailed descriptions are contained in Table 25-2. Some signals (the MECC bus for example) are additionally described in other chapters, but are described here also for completeness, with emphasis on their debugging utility.

**Table 25-2. Debug, Watchpoint and Test Signal Summary**

| Name | Description | Functional Block | Function | Reset Value | I/O | Page # |
|---|---|---|---|---|---|---|
| MDVAL | Memory data-valid | Debug | Selectable data-valid signal from either DDR SDRAM controller or LBC. | 1 | O | 25-6 |
| MECC[0:7] | DDR error correcting code | DDR SDRAM | In debug mode, the high-order six bits carry debug information (transaction source ID and data-valid indication). | 0x08 | O[1] | 25-7 |
| MSRCID[0:1] | Memory source ID | Debug | Selectable transaction source ID from either DDR SDRAM controller or local bus controller. | Reset_cfg | O | 25-7 |
| MSRCID[2:4] | | | | 111 | O | 25-7 |
| TRIG_IN | Trigger in | Debug | Trigger for various function in the watchpoint monitor and trace buffer. | 1 | I | 25-7 |
| TRIG_OUT | Trigger out | Debug | Can be used externally for triggering a logic analyzer. Additionally, it can be used for observing system ready indication. Functions are multiplexed onto this signal depending on TOSR[SEL] (see Table 25-25). | 1 | O | 25-7 |
| TCK | Test clock | Debug | Clock for JTAG testing. Internally pulled up. | 1 | I | 25-8 |

Table 25-2. Debug, Watchpoint and Test Signal Summary (continued)

| Name | Description | Functional Block | Function | Reset Value | I/O | Page # |
|---|---|---|---|---|---|---|
| TDI | Test data input | Debug | Serial input for instructions and data to the JTAG test subsystem. Internally pulled up. | 1 | I | 25-8 |
| TDO | Test data output | Debug | Serial data output for the JTAG test subsystem. High impedance except when scanning out data. | Hi Z | O | 25-8 |
| TMS | Test mode select | Debug | Carries commands to the TAP controller for boundary scan operations. Internally pulled up. | 1 | I | 25-8 |
| $\overline{\text{TRST}}$ | Test reset | Debug | Resets the TAP controller asynchronously. | — | I | 25-8 |
| THERM[0:1] | Thermal resistor access | Test | These pins tie directly to an internal resistor whose value varies linearly with temperature. | — | I | 25-8 |
| $\overline{\text{TEST\_SEL}}$ | Test select 1 | Test | Factory test. Must be negated (pulled high) for normal operation. | — | I | 25-8 |
| $\overline{\text{LSSD\_MODE}}$ | Test | Test | Factory Test. Refer to the *MPC8536E Integrated Processor Hardware Specifications* for proper treatment. | | I | 25-8 |
| L1_TSTCLK | Test | Test | Factory Test. Refer to the *MPC8536E Integrated Processor Hardware Specifications* for proper treatment. | | I | 25-8 |
| L2_TSTCLK | Test | Test | Factory Test. Refer to the *MPC8536E Integrated Processor Hardware Specifications* for proper treatment. | | I | 25-8 |

[1] While these signals are normally bidirectional, when sourcing debug information they are output only.

## 25.2.2 Detailed Signal Descriptions

This section describes the details of the debug, watchpoint monitor, and JTAG test signals

### 25.2.2.1 Debug Signals—Details

Table 25-3 describes all signals associated with device debug modes.

Table 25-3. Debug Signals—Detailed Signal Descriptions

| Signal | I/O | Description | |
|---|---|---|---|
| MDVAL | O | Memory data-valid. Indicates when valid data is available. May be used by a logic analyzer to capture the data on the data bus. | |
| | | **State Meaning** | Asserted—Indicates that data is valid on the data bus during the current clock cycle. When the DDR SDRAM interface is selected to source information on MDVAL, this signal is valid for every cycle that data is driven or received on the DDR SDRAM interface. When the LBC is selected, this signal is valid for every cycle that data is driven or received on the local bus interface. The assertion of this signal may be used by a logic analyzer to capture data. |
| | | **Timing** | Asserted/Negated—Referenced to the selected interface, (DDR or local bus). Asserts when data is valid. Assertions are held for the duration of the transfer. Read data timing is similar to MA. Write data timing is similar to the output MDQ. |

Table 25-3. Debug Signals—Detailed Signal Descriptions (continued)

| Signal | I/O | Description | |
|---|---|---|---|
| MECC[0:7] | O | Memory ECC. DDR error checking and correcting. The normally bidirectional operation of the memory ECC (MECC) bus is described in Section 8.5.11, "Error Checking and Correcting (ECC)." This bus is used for debug functions when MSRCID1 is sampled low during POR.<br>In debug mode, the high-order 5 bits (MECC[0:4]) may be used to provide the transaction source ID and MECC5 can be used as the data-valid indicator.<br>In debug mode, MECC[0:5] is constantly driven with debug information and must be disconnected from the DDR memory's ECC pins. | |
| | | **State Meaning** | Asserted/Negated—In debug mode, MECC[0:5] is always driven. The source ID values appear during RAS and CAS cycles. A value of 0x1F (all ones) is driven during cycles other than RAS and CAS. The data-valid indicator appears when data is being received or driven on the pins. |
| | | **Timing** | Driven every cycle in debug mode. |
| MSRCID[0:4] | O | Memory source ID. Attribute signals associated with the memory interface that indicate the source ID for a transaction on an SDRAM interface. The SDRAM interface, DDR or local bus, to which the debug information applies is specified during POR with MSRCID0 as shown in Table 25-1. Two of these signals serve as reset configuration input signals. | |
| | | **State Meaning** | Asserted/Negated—In debug mode, always driven with the value of the source ID. The source ID has a value of 0x1F for cycles other than RAS and CAS. The encodings shown in Table 25-26 provide detailed information about a memory transaction. |
| | | **Timing** | Driven every cycle in debug mode. Similar timing to MA. |

### 25.2.2.2 Watchpoint Monitor Trigger Signals—Details

Table 25-4 shows detailed descriptions of the watchpoint monitor and trace buffer signals.

Table 25-4. Watchpoint and Trigger Signals—Detailed Signal Descriptions

| Signal | I/O | Description | |
|---|---|---|---|
| TRIG_IN | I | Trigger in. Can be used to trigger the watchpoint and trace buffers. Note this is an active-high (rising-edge triggered) signal. | |
| | | **State Meaning** | Asserted—Indicates that a programmed/armed external event has been detected. Assertion may be used internally to trigger trace buffers and watchpoint mechanisms. |
| | | **Timing** | Assertion/Negation—The MPC8536E interprets TRIG_IN as asserted on detection of the rising edge. It may occur at any time. Must remain asserted for at least 3 system clocks to be recognized internally. |

Table 25-4. Watchpoint and Trigger Signals—Detailed Signal Descriptions

| Signal | I/O | Description | |
|---|---|---|---|
| TRIG_OUT | O | Trigger out. Function determined by TOSR[SEL]. When TOSR[SEL] is non-zero, it can be used for triggering external devices, like a logic analyzer, with either the watchpoint monitor, the trace buffer, or the performance monitor as trigger sources. When TOSR[SEL] is cleared, TRIG_OUT is multiplexed with READY, which indicates the operational readiness of the device (running or in low-power or debug modes). See Chapter 4, "Reset, Clocking, and Initialization," and Chapter 23, "Global Utilities," for more details about reset, low-power, and debug states. | |
| | | **State Meaning** | Asserted—When TOSR[SEL] is all zeros, serves as the READY signal, indicating that the device is not in a low-power or debug mode and that it has emerged from reset. SEL ≠ 0 indicates that a programmed trigger event has occurred.<br>Negation—No final watchpoint match condition |
| | | **Timing** | Assertion may occur at any time. Remains asserted for at least 3 system clocks |

## 25.2.2.3 Test Signals—Details

Table 25-5 shows detailed descriptions of the JTAG test signals.

Table 25-5. JTAG Test and Other Signals—Detailed Signal Descriptions

| Signal | I/O | Description | |
|---|---|---|---|
| TCK | I | JTAG test clock. | |
| | | **State Meaning** | Asserted/Negated—Should be driven by a free-running clock signal with a 30–70% duty cycle. Input signals to the TAP are clocked in on the rising edge. Changes to the TAP output signals occur on the falling edge. The test logic allows TCK to be stopped. An unterminated input appears as a high signal level to the test logic due to an internal pull-up resistor. |
| | | **Timing** | See IEEE 1149.1 standard for more details. |
| TDI | I | JTAG test data input. | |
| | | **State Meaning** | Asserted/Negated—The value present on the rising edge of TCK is clocked into the selected JTAG test instruction or data register. An unterminated input appears as a high signal level to the test logic due to an internal pull-up resistor. |
| | | **Timing** | See IEEE 1149.1 standard for more details. |
| TDO | O | JTAG test data output. | |
| | | **State Meaning** | Asserted/Negated—The contents of the selected internal instruction or data register are shifted out on this signal on the falling edge of TCK. Remains in a high-impedance state except when scanning data. |
| | | **Timing** | See IEEE 1149.1 standard for more details. |
| TMS | I | JTAG test mode select. | |
| | | **State Meaning** | Asserted/Negated—Decoded by the internal JTAG TAP controller to distinguish the primary operation of the test support circuitry. An unterminated input appears as a high signal level to the test logic due to an internal pull-up resistor. |
| | | **Timing** | See IEEE 1149.1 standard for more details. |

Table 25-5. JTAG Test and Other Signals—Detailed Signal Descriptions

<table>
<tr><th>Signal</th><th>I/O</th><th colspan="2">Description</th></tr>
<tr><td rowspan="3">$\overline{\text{TRST}}$</td><td rowspan="3">I</td><td colspan="2">JTAG test reset.</td></tr>
<tr><td><b>State Meaning</b></td><td>Asserted—Causes asynchronous initialization of the internal JTAG TAP controller. Must be asserted during power-on reset in order to properly initialize the JTAG TAP and for normal operation of the MPC8536E. An unterminated input appears as a high signal level to the test logic due to an internal pull-up resistor.<br>Negated— Normal operation.</td></tr>
<tr><td><b>Timing</b></td><td>See IEEE 1149.1 standard for more details.</td></tr>
<tr><td>$\overline{\text{LSSD\_MODE}}$</td><td>I</td><td colspan="2">Used for factory test. Refer to the <i>MPC8536E Integrated Processor Hardware Specifications</i> for proper treatment.</td></tr>
<tr><td>L1_TSTCLK</td><td>I</td><td colspan="2">Used for factory test. Refer to the <i>MPC8536E Integrated Processor Hardware Specifications</i> for proper treatment.</td></tr>
<tr><td>L2_TSTCLK</td><td>I</td><td colspan="2">Used for factory test. Refer to the <i>MPC8536E Integrated Processor Hardware Specifications</i> for proper treatment.</td></tr>
<tr><td>THERM[0:1]</td><td>I</td><td colspan="2">These signals provide access to an internal resistor that has a value that varies linearly with temperature. The actual value for the resistor varies from device to device, but the linear relationship between temperature and resistance is consistent. See the <i>Integrated Processor Hardware Specifications</i> for more information on how to accurately measure the junction temperature of a device. Note that this thermal resistor is intended for engineering development only.</td></tr>
<tr><td>$\overline{\text{TEST\_SEL}}$</td><td>I</td><td colspan="2">Used for factory test. Should be negated (pulled high) for normal operation.</td></tr>
</table>

## 25.3 Memory Map/Register Definition

Table 25-6 shows the memory-mapped debug and watchpoint registers of the MPC8536E. Undefined 4-byte address spaces within offset 0x000–0xFFF are reserved.

In this table and in the register figures and field descriptions, the following access definitions apply:

- Reserved fields are always ignored for the purposes of determining access type.
- R/W, R, and W (read/write, read only, and write only) indicate that all the non-reserved fields in a register have the same access type.
- w1c indicates that all of the non-reserved fields in a register are cleared by writing ones to them.
- Mixed indicates a combination of access types.
- Special is used when no other category applies. In this case the register figure and field description table should be read carefully.

Table 25-6. Debug and Watchpoint Monitor Memory Map

<table>
<tr><th>Local Memory Offset</th><th>Register</th><th>Access</th><th>Reset</th><th>Section/Page</th></tr>
<tr><td colspan="5"><b>Watchpoint Monitor Registers</b></td></tr>
<tr><td>0xE_2000</td><td>WMCR0—Watchpoint monitor control register 0</td><td>R/W</td><td>0x0000_0000</td><td>25.3.1.1/25-10</td></tr>
<tr><td>0xE_2004</td><td>WMCR1—Watchpoint monitor control register 1</td><td>R/W</td><td>0x0000_0000</td><td>25.3.1.1/25-10</td></tr>
</table>

**Table 25-6. Debug and Watchpoint Monitor Memory Map (continued)**

| Local Memory Offset | Register | Access | Reset | Section/Page |
|---|---|---|---|---|
| 0xE_200C | WMAR—Watchpoint monitor address register | R/W | 0x0000_0000 | 25.3.1.2/25-12 |
| 0xE_2014 | WMAMR—Watchpoint monitor address mask register | R/W | 0x0000_0000 | 25.3.1.3/25-13 |
| 0xE_2018 | WMTMR—Watchpoint monitor transaction mask register | R/W | 0x0000_0000 | 25.3.1.4/25-13 |
| 0xE_201C | WMSR—Watchpoint monitor status register | R/W | 0x0000_0000 | 25.3.1.5/25-15 |
| **Trace Buffer Registers** | | | | |
| 0xE_2040 | TBCR0—Trace buffer control register 0 | R/W | 0x0000_0000 | 25.3.2.1/25-15 |
| 0xE_2044 | TBCR1—Trace buffer control register 1 | R/W | 0x0000_0000 | 25.3.2.1/25-15 |
| 0xE_204C | TBAR—Trace buffer address register | R/W | 0x0000_0000 | 25.3.2.2/25-18 |
| 0xE_2054 | TBAMR—Trace buffer address mask register | R/W | 0x0000_0000 | 25.3.2.3/25-18 |
| 0xE_2058 | TBTMR—Trace buffer transaction mask register | R/W | 0x0000_0000 | 25.3.2.4/25-19 |
| 0xE_205C | TBSR—Trace buffer status register | R/W | 0x0000_0000 | 25.3.2.5/25-19 |
| 0xE_2060 | TBACR—Trace buffer access control register | R/W | 0x0000_0000 | 25.3.2.6/25-20 |
| 0xE_2064 | TBADHR—Trace buffer access data high register | R/W | 0x0000_0000 | 25.3.2.7/25-21 |
| 0xE_2068 | TBADR—Trace buffer access data register | R/W | 0x0000_0000 | 25.3.2.8/25-21 |
| **Context ID Registers** | | | | |
| 0xE_20A0 | PCIDR—Programmed context ID register | R/W | 0x0000_0000 | 25.3.3.1/25-22 |
| 0xE_20A4 | CCIDR—Current context ID register | R/W | 0x0000_0000 | 25.3.3.2/25-23 |
| **Other Registers** | | | | |
| 0xE_20B0 | TOSR—Trigger output source register | R/W | 0x0000_0000 | 25.3.4.1/25-23 |

## 25.3.1 Watchpoint Monitor Register Descriptions

The following sections describe the control registers for the watchpoint monitor facility.

### 25.3.1.1 Watchpoint Monitor Control Registers 0–1 (WMCR0, WMCR1)

The watchpoint monitor control registers (WMCR0, WMCR1) shown in Figure 25-2 and Figure 25-3 control the specification of watchpoint monitor events.

Offset 0x000 Access: Read/Write

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 20 | 21 23 | 24 31 |
|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | EN | AMD | TMD | ECEN | NECEN | SIDEN | TIDEN | — | STRT | — |
| Reset | All zeros | | | | | | | | | |

**Figure 25-2. Watchpoint Monitor Control Register 0 (WMCR0)**

Table 25-7 describes WMCR0 fields.

**Table 25-7. WMCR0 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | EN | Enable<br>0 Watchpoint monitor events are not flagged.<br>1 A watchpoint monitor event is flagged. |
| 1 | AMD | Address match disable. Qualifies address match as a watchpoint event criterion.<br>0 Address matching is used to recognize a watchpoint event.<br>1 Address matching does not affect watchpoint event detection. |
| 2 | TMD | Transaction match disable. Qualifies transaction type match (as defined in WMCR1[IFSEL] and WMTMR) as a watchpoint event criterion.<br>0 A transaction type match is used to recognize watchpoint events.<br>1 A transaction type match does not affect watchpoint event detection. |
| 3 | ECEN | Equal context enable. Qualifies the matching of current context with programmed context as a watchpoint event criterion, as written in the context registers described in Section 25.3.3, “Context ID Registers.”<br>0 Current context match does not affect watchpoint event detection.<br>1 Watchpoint events are qualified by comparing current context with the programmed context event value.<br>**Note:** ECEN and NECEN must not be enabled in the same run. If both are set, watchpoint events are inhibited (never occur). |
| 4 | NECEN | Not equal context enable. Qualifies the matching of current context with programmed context as a watchpoint event criterion, as written in the context registers described in Section 25.3.3, “Context ID Registers.”<br>0 The failure of a current context match does not affect watchpoint event detection<br>1 Watchpoint events are qualified with NOT getting a current context compare with the programmed context event value.<br>**Note:** ECEN and NECEN must not be enabled in the same run. If both are set, watchpoint events are inhibited (never occur). |
| 5 | SIDEN | Source ID enable<br>0 Source ID does not affect watchpoint event detection.<br>1 Watchpoint events are qualified by comparison with the programmed WMCR1(SID) value. |
| 6 | TIDEN | Target ID enable<br>0 Target ID does not affect watchpoint event detection.<br>1 Watchpoint events are qualified by comparison with the programmed WMCR1(TID) value. |
| 7–20 | — | Reserved |
| 21–23 | STRT | Start condition. Specifies the event that arms the watchpoint monitor to start looking for the programmed event.<br>000 No event. Armed immediately<br>001 Trace buffer event is detected<br>010 Performance monitor signals overflow<br>011 TRIG_IN transitions from 0 to 1<br>100 TRIG_IN transitions from 1 to 0<br>101 Current context ID equals programmed context ID<br>110 Current context ID is not equal to programmed context ID<br>111 Reserved |
| 24–31 | — | Reserved |

Figure 25-3 shows WMCR1.

Offset 0x004 Access: Read/Write

| | 0–4 | 5–7 | 8–10 | 11–15 | 16–26 | 27–31 |
|---|---|---|---|---|---|---|
| R<br>W | — | IFSEL | — | SID | — | TID |
| Reset | All zeros | | | | | |

**Figure 25-3. Watchpoint Monitor Control Register 1 (WMCR1)**

Table 25-8 describes the WMCR1 fields.

**Table 25-8. WMCR1 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–4 | — | Reserved |
| 5–7 | IFSEL | Interface selection. Selects the address, transaction type (as defined in WMTMR), and other attributes to be used for comparison<br>000 Selects e500 coherency module (ECM) dispatch interface<br>001 Selects internal DDR SDRAM interface<br>010 Selects internal PCI outbound interface<br>011 Reserved<br>100 Selects internal PCI Express 1 outbound interface<br>101 Selects internal PCI Express 2 outbound interface<br>110 Selects internal PCI Express 3 outbound interface<br>111 Reserved |
| 8–10 | — | Reserved |
| 11–15 | SID | Source ID. Specifies the source ID associated with WMCR0[SIDEN]. For a definition of the source ID, see Table 25-26. |
| 16–26 | — | Reserved |
| 27–31 | TID | Target ID. Specifies the target ID associated with WMCR0[TIDEN]. For a definition of the target ID see Table 25-26. |

## 25.3.1.2 Watchpoint Monitor Address Register (WMAR)

The watchpoint monitor address register (WMAR) shown in Figure 25-4 contains the address to match against if WMCR[AMD] is clear. Note that this address may be further qualified with the bits described in Section 25.3.1.3, "Watchpoint Monitor Address Mask Register (WMAMR)."

Table 25-9 describes the WMAR fields.

Offset 0x00C Access: Read/Write

| | 0–31 |
|---|---|
| R<br>W | WMA |
| Reset | All zeros |

**Figure 25-4. Watchpoint Monitor Address Register (WMAR)**

Table 25-9. WMAR Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–31 | WMA | Watchpoint monitor address. |

### 25.3.1.3 Watchpoint Monitor Address Mask Register (WMAMR)

The watchpoint monitor address mask register (WMAMR) shown in Figure 25-5 contains the mask for the address in the WMAR.

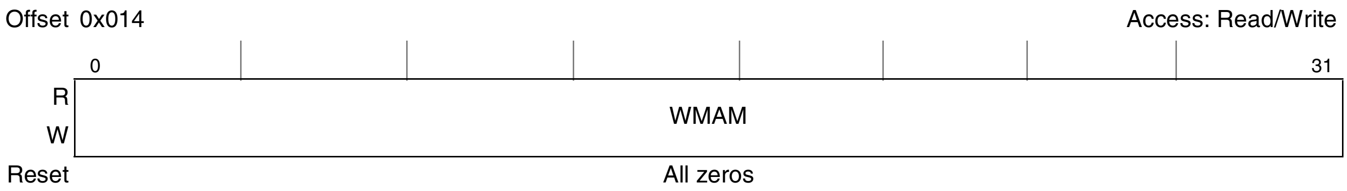


Figure 25-5. Watchpoint Monitor Address Mask Register (WMAMR)

Table 25-10 describes the WMAMR fields.

Table 25-10. WMAMR Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–31 | WMAM | Watchpoint monitor address mask. A value of zero masks the address comparison for the corresponding address bit. These bits only mask the address bits generated by the hardware, but do not affect the bits specified in WMAR. A bit that is masked from the comparison should be set to 0 in WMAR. |

### 25.3.1.4 Watchpoint Monitor Transaction Mask Register (WMTMR)

The watchpoint monitor transaction mask register (WMTMR), shown in Figure 25-6, specifies which transaction types to monitor. WMTMR allows users to qualify watchpoint events specifically with any combination of transaction types. As shown in Table 25-11, each bit represents as many as four separate transaction types; one for each interface. Setting a bit enables watchpoint monitoring for the corresponding transaction types.

Because the supported transaction types vary by interface, the type designated by a WMTMR field also depends on the interface specified by WMCR1[IFSEL]. Table 25-12 lists transaction types associated with each WMTMR bit by interface.

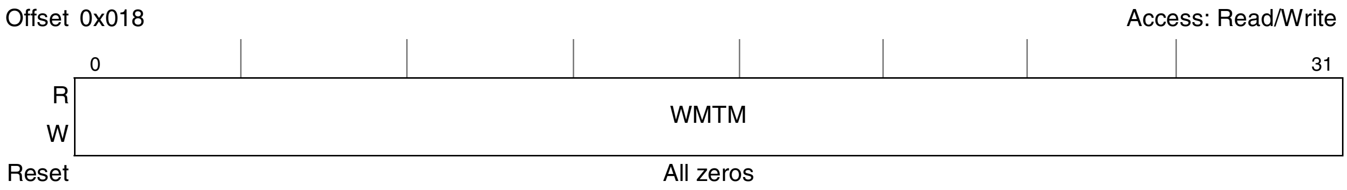


Figure 25-6. Watchpoint Monitor Transaction Mask Register (WMTMR)

Table 25-11 describes the WMTMR fields.

**Table 25-11. WMTMR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–31 | WMTM | Watchpoint monitor transaction mask. Each bit corresponds to a transaction type as defined in Table 25-12. The transaction associated with any particular bit may be different depending on the interface being monitored. A value of 1 for a given mask bit enables the matching of the transaction associated with that bit. These bits are meaningful only when WMCR0[TMD]=0. |

The following table, Table 25-12, defines the transactions associated with each transaction mask bit for the different interfaces supported by the watchpoint monitor.

**Table 25-12. Transaction Types By Interface**

| Bit | Description | | | |
|---|---|---|---|---|
| | e500 Coherency Module Dispatch | DDR Controller | PCI Outbound Request | PCI Express Outbound Transaction |
| 0 | Write with local processor snoop | Write | Memory write | Posted Write |
| 1 | Write with no local processor snoop | — | I/O write | Non-posted Write |
| 2 | Write with allocate(L2 stashing) | Write with allocate | — | — |
| 3 | Write with allocate and lock (L2 stashing with locking) | Write with allocate and lock | — | — |
| 4 | Reserved | — | — | — |
| 5 | Reserved | — | — | — |
| 6 | Reserved | — | — | — |
| 7 | Reserved | — | — | — |
| 8 | Read with local processor snoop | Read | Memory Read | Read |
| 9 | Read with no local processor snoop | — | I/O Read | — |
| 10 | Read with unlock | Read with unlock | — | — |
| 11 | Reserved | — | — | Read Response |
| 12 | Reserved | — | — | — |
| 13–15 | Reserved | — | — | — |
| 16 | ATOMIC clear | ATOMIC clear | — | — |
| 17 | ATOMIC set | ATOMIC set | — | — |
| 18 | ATOMIC decrement | ATOMIC decrement | — | — |
| 19 | ATOMIC increment | ATOMIC increment | — | — |
| 20–24 | Reserved | — | — | — |
| 25 | Address only transaction | — | — | — |
| 26–31 | Reserved | — | — | — |

### 25.3.1.5 Watchpoint Monitor Status Register (WMSR)

The watchpoint monitor status register (WMSR) shown in Figure 25-7 indicates the state of the watchpoint monitor.

Offset 0x01C Access: Read/Write

| | 0 | 1 | 2–31 |
|---|---|---|---|
| R<br>W | ACT | TRIG | — |
| Reset | All zeros | | |

**Figure 25-7. Watchpoint Monitor Status Register (WMSR)**

Table 25-13 describes the WMSR fields.

**Table 25-13. WMSR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | ACT | Active<br>0 The start triggering event has not occurred; watchpoint monitor is not armed.<br>1 The start triggering event has occurred; watchpoint monitor is armed. |
| 1 | TRIG | Triggered<br>0 The programmed event in WMCR0 has not yet been triggered.<br>1 The programmed event in WMCR0 has been triggered at least once. |
| 2–31 | — | Reserved |

## 25.3.2 Trace Buffer Register Descriptions

The following sections describes the trace buffer registers.

### 25.3.2.1 Trace Buffer Control Registers (TBCR0, TBCR1)

The trace buffer control registers (TBCR0, TBCR1), shown in Figure 25-8 and Figure 25-9, specify trace buffer events.

Offset 0x040 Access: Read/Write

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8–13 | 14–15 | 16–20 | 21–23 | 24–28 | 29–31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | EN | AMD | TMD | ECEN | NECEN | SIDEN | TIDEN | HALT | — | MODE | — | STRT | — | STOP |
| Reset | All zeros | | | | | | | | | | | | | |

**Figure 25-8. Trace Buffer Control Register 0 (TBCR0)**

Table 25-14 describes the TBCR0 fields.

**Table 25-14. TBCR0 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0 | EN | Enable<br>0 The trace buffer facility is disabled.<br>1 The trace buffer facility is enabled. |
| 1 | AMD | Address match disable<br>0 The address match is used to qualify a trace buffer event.<br>1 The address match is ignored when detecting a trace buffer event. |
| 2 | TMD | Transaction match disable<br>0 The transaction type match is used to qualify a trace buffer event.<br>1 The transaction type match is ignored when detecting a trace buffer event. |
| 3 | ECEN | Equal context enable. Qualifies the matching of current context with programmed context as a trace buffer event criterion, as written in the context registers described in Section 25.3.3, "Context ID Registers."<br>0 Current context match does not affect trace buffer event detection<br>1 Trace buffer events are qualified by comparing current context with the programmed context event value.<br>**Note:** ECEN and NECEN must not be enabled in the same run. If both are set, watchpoint events are inhibited (never occur). |
| 4 | NECEN | Not equal context enable. Qualifies the matching of current context with programmed context as a trace buffer event criterion, as written in the context registers described in Section 25.3.3, "Context ID Registers."<br>0 The failure of a current context match does not affect trace buffer event detection<br>1 trace buffer events are qualified with NOT getting a current context compare with the programmed context event value.<br>**Note:** ECEN and NECEN must not be enabled in the same run. If both are set, watchpoint events are inhibited (never occur). |
| 5 | SIDEN | Source ID enable<br>0 Trace buffer events ignore the programmed source ID value.<br>1 Trace buffer events are qualified by comparison with the programmed SID event value. |
| 6 | TIDEN | Target ID enable<br>0 Trace buffer events ignore the programmed TID event value.<br>1 Trace buffer events are qualified by comparison with the programmed TID event value. This comparison only applies when the ECM is selected for tracing (TBCR1[IFSEL] is all zeros). |
| 7 | HALT | Halt causes the trace buffer to stop tracing immediately. TBSR[ACT] remains set when this bit is set. |
| 8–13 | — | Reserved |
| 14–15 | MODE | Trace mode. Specifies one of two trace modes.<br>00 Trace every valid transaction<br>01 Reserved<br>10 Trace only cycles in which a trace event is detected. Note that if EN and other TBCR0 fields are not properly programmed to specify a traceable event, tracing occurs for every valid address.<br>11 Reserved |
| 16–20 | — | Reserved |

**Table 25-14. TBCR0 Field Descriptions (continued)**

| Bits | Name | Description |
|---|---|---|
| 21–23 | STRT | Start condition. Specifies the event that arms the trace buffer to start looking for the programmed event<br>000 No event. Armed immediately<br>001 Watchpoint monitor event is detected<br>010 Trace buffer event is detected<br>011 Performance monitor signals overflow<br>100 TRIG_IN transitions from 0 to 1<br>101 TRIG_IN transitions from 1 to 0<br>110 Current context ID equals programmed context ID<br>111 Current context ID does not equal programed context ID |
| 24–28 | — | Reserved |
| 29–31 | STOP | Trace stop mode. Specifies the event that stops the updating of the trace buffer after it has been started. Trace buffer only stops after it has been triggered at least once.<br>000 Buffer is full<br>001 Watchpoint monitor event is detected<br>010 Trace buffer event is detected<br>011 Performance monitor signals overflow<br>100 TRIG_IN transitions from 0 to 1<br>101 TRIG_IN transitions from 1 to 0<br>110 Current context ID equals programmed context ID<br>111 Current context ID does not equal programed context ID |

Offset 0x044 Access: Read/Write

| | 0 4 | 5 7 | 8 10 | 11 15 | 16 26 | 27 31 |
|---|---|---|---|---|---|---|
| R<br>W | — | IFSEL | — | SID | — | TID |
| Reset | All zeros | | | | | |

**Figure 25-9. Trace Buffer Control Register 1 (TBCR1)**

Table 25-15 describes the TBCR1 fields.

**Table 25-15. TBCR1 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–4 | — | Reserved |
| 5–7 | IFSEL | Interface selection. Specifies the interface that sources information for both comparison/buffer control and buffer data capture.<br>000 Selects e500 coherency module (ECM) dispatch interface<br>001 Selects internal DDR SDRAM interface<br>010 Selects internal PCI outbound interface<br>011 Reserved<br>100 Selects internal PCI Express 1 outbound interface<br>101 Selects internal PCI Express 2 outbound interface<br>110 Selects internal PCI Express 3 outbound interface<br>111 Reserved |
| 8–10 | — | Reserved |
| 11–15 | SID | Source ID. Specifies the source ID associated with TBCR0[SIDEN]. The source ID is defined in Table 25-26. |

**Table 25-15. TBCR1 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 16–26 | — | Reserved |
| 27–31 | TID | Target ID. Specifies the target ID associated with TBCR0[TIDEN]. The target ID is defined in Table 25-26. |

### 25.3.2.2 Trace Buffer Address Register (TBAR)

The trace buffer address register (TBAR) shown in Figure 25-10 contains the address to match against (if TBCR0[AMD] is zero). This address may be further qualified by the mask bits defined in Section 25.3.2.3, "Trace Buffer Address Mask Register (TBAMR)."

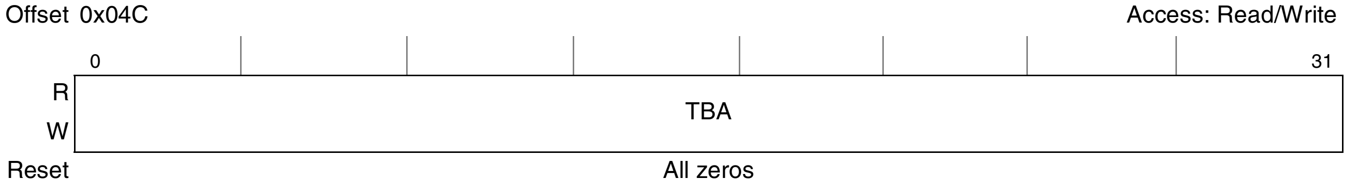


**Figure 25-10. Trace Buffer Address Register (TBAR)**

Table 25-16 describes the TBAR field.

**Table 25-16. TBAR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–31 | TBA | Trace buffer address. |

### 25.3.2.3 Trace Buffer Address Mask Register (TBAMR)

The trace buffer address mask register (TBAMR) shown in Figure 25-11 contains a mask for the TBAR, which allows excluding address bits from the comparison.

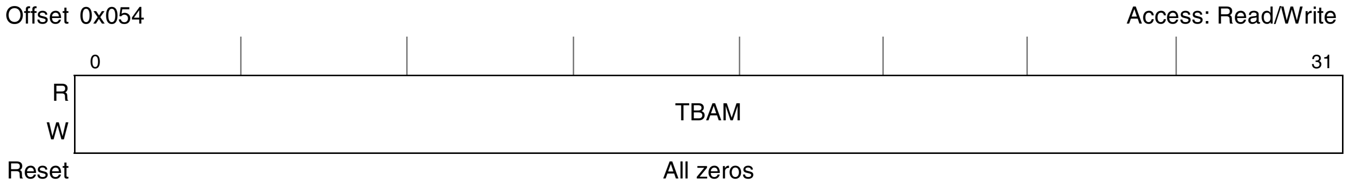


**Figure 25-11. Trace Buffer Address Mask Register (TBAMR)**

Table 25-17 describes the TBAMR field.

**Table 25-17. TBAMR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–31 | TBAM | Trace buffer address mask.A value of zero masks the address comparison for the corresponding address bit. These bits only mask the address bits generated by the hardware, but do not affect the bits specified in TBAR. A bit that is masked from the comparison should be set to 0 in TBAR. |

### 25.3.2.4 Trace Buffer Transaction Mask Register (TBTMR)

The trace buffer transaction mask register (TBTMR) shown in Figure 25-12 specifies which transaction types to monitor. Each bit in the TBTMR represents a transaction type on the selected interface. The transaction associated with any particular bit depends on the interface being monitored as specified by TBCR1[IFSEL]. Note that the transactions used for defining trace buffer events are the same as those defined for watchpoint monitor events. Thus, Table 25-12 defines the transaction types associated with each interface. Setting a bit enables a hit when this transaction is matched (provided all other match criteria are met and TBCR0[TMD] is clear).

Different interfaces support different transaction types, and the same bit may represent different transaction types depending on the interface.

Offset 0x058 Access: Read/Write

| | 0–31 |
|---|---|
| R<br>W | TBTM |
| Reset | All zeros |

**Figure 25-12. Trace Buffer Transaction Mask Register (TBTMR)**

Table 25-18 describes the TBTMR field.

**Table 25-18. TBTMR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–31 | TBTM | Trace buffer transaction mask. Each bit corresponds to a transaction type as defined in Table 25-12. The transaction associated with a bit depends on the interface being monitored. A value of 1 for a given mask bit enables the matching of the transaction associated with that bit. These bits are meaningful only when TBCR0[TMD]=0. |

### 25.3.2.5 Trace Buffer Status Register (TBSR)

The trace buffer status register (TBSR) shown in Figure 25-13 indicates the operational state of the trace buffer.

Offset 0x05C Access: Read/Write

| | 0 | 1 | 2 | 3 | 4–23 | 24–31 |
|---|---|---|---|---|---|---|
| R<br>W | ACT | TRIG | STP | WRAP | — | C_INDX |
| Reset | All zeros | | | | | |

**Figure 25-13. Trace Buffer Status Register (TBSR)**

Table 25-19 describes the TBSR fields.

Table 25-19. TBSR Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0 | ACT | Active. Indicates trace buffer activity.<br>0 The start triggering event has not yet occurred. Trace buffer is not armed.<br>1 The start triggering event has occurred. Trace buffer is armed. |
| 1 | TRIG | Triggered. Indicates whether or not a programmed event has been triggered.<br>0 The programmed event in TBCR0 has not yet been triggered.<br>1 The programmed event in TBCR0 has been triggered at least once. |
| 2 | STP | Stopped. Indicates whether or not a trace buffer stop condition has been detected.<br>0 No stop condition yet detected.<br>1 The trace buffer has detected a stop condition and is no longer capturing events. |
| 3 | WRAP | Wrapped. Indicates that the trace buffer write pointer has wrapped to the beginning of the buffer at least once. Set when the last entry of the trace buffer is written.<br>0 Pointer has not yet wrapped.<br>1 Pointer has wrapped to the beginning at least once. |
| 4–23 | — | Reserved |
| 24–31 | C_INDX | Current index. Represents the current value of the write pointer at the time TBSR was read. This value may be written by software to initialize the write pointer; however, software is not allowed to write the write pointer while the trace buffer is active. Writes are ignored while the trace buffer is active. It is recommended to write the status register before enabling the trace buffer in order to zero out any bits that might have been set during a prior run and to initialize the write pointer to zero. |

## 25.3.2.6 Trace Buffer Access Control Register (TBACR)

The trace buffer access control register (TBACR) enables software to read or write the trace buffer. Each entry is 64 bits; therefore, it takes one write of TBACR and two reads of the access data register (TBADR and TBADHR) to read one 256-entry array entry. Similarly, it takes one write of TBACR and two writes of TBADR and TBADHR to write one array entry. Software can access any entry by writing the appropriate index into TBACR[INDX]. To read or write the buffer sequentially, starting with entry 0, the index must start with a value of 0 and increment every time a new entry is accessed.

TBACR is shown in Figure 25-14.

Offset 0x060 Access: Read/Write

| | 0 | 1 | 2–23 | 24–31 |
|---|---|---|---|---|
| R<br>W | RD | WR | — | INDX |
| Reset | All zeros | | | |

Figure 25-14. Trace Buffer Access Control Register (TBACR)

Table 25-20 describes the TBACR fields.

Table 25-20. TBACR Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0 | RD | Read command. When set, a trace buffer read is performed using the value of TBACR[INDX]. This bit is automatically cleared when the read is performed. |
| 1 | WR | Write command. When set, a trace buffer write is performed using the value of TBACR[INDX]. This bit is automatically cleared when the write is performed. A write occurs only if the trace buffer is not active: write requests are ignored while the buffer is active. |
| 2–23 | — | Reserved |
| 24–31 | INDX | Buffer index to read from or write into (0–255). Used in conjunction with TBACR[RD] and TBACR[WR]. |

### 25.3.2.7 Trace Buffer Access Data High Register (TBADHR)

The trace buffer access data high register (TBADHR), shown in Figure 25-15, contains the high-order 32 bits of the data read from the trace buffer during a software-initiated read command (TBACR[RD]), or the write data to be written into the trace buffer during a software-initiated write command (TBACR[WR]). TBACR must be configured to perform a read before this register contains valid data. This register must be initialized by software before configuring the TBACR to perform a write command.

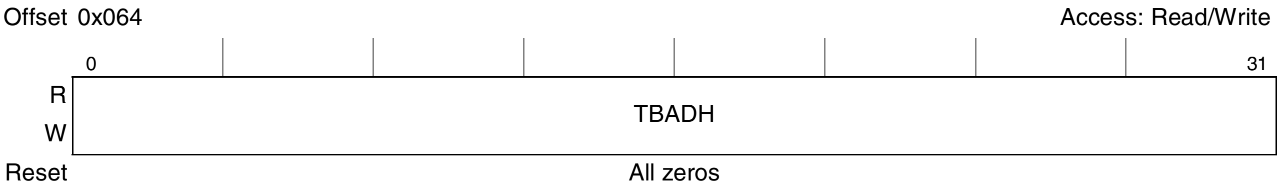


Figure 25-15. Trace Buffer Read High Register (TBADHR)

Table 25-21 describes TBADHR.

Table 25-21. TBADHR Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–31 | TBADH | Trace buffer access data high. The higher 32 bits of the data read from or to be written into the trace buffer, depending on whether the array is accessed with a read or a write. |

### 25.3.2.8 Trace Buffer Access Data Register (TBADR)

The trace buffer access data register (TBADR), shown in Figure 25-16, contains the low-order 32 bits of the data read from the trace buffer during a software-initiated read command (TBACR[RD]) or the write data to be written into the trace buffer during a software-initiated write command (TBACR[WR]). TBACR must be configured to perform a read before this register contains valid data. This register must be initialized by software before configuring the TBACR to perform a write command.

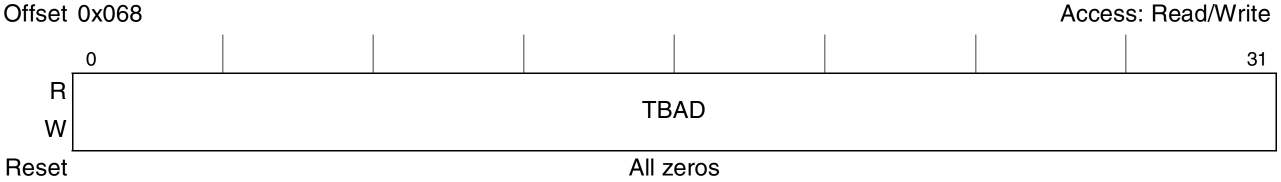


**Figure 25-16. Trace Buffer Access Data Register (TBADR)**

Table 25-22 describes the TBADR field.

**Table 25-22. TBADR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–31 | TBAD | Trace buffer access data. Corresponds to the lower 32 bits of the data read from the trace buffer or to be written into the trace buffer, depending on whether software is accessing the array with a read or a write. |

## 25.3.3 Context ID Registers

This section describes the context ID registers. The current context ID register (CCIDR) and programmed context ID registers (PCIDR) are set by software and facilitate debugging complex software.

### 25.3.3.1 Programmed Context ID Register (PCIDR)

The programmed context ID register (PCIDR), shown in Figure 25-17, contains the user-programmed context ID. This register can be configured to trigger watchpoint events when its value matches the current context ID register (CCIDR), as controlled by WMCR0[ECEN] and WMCR0[NECEN]. See Section 25.3.1.1, “Watchpoint Monitor Control Registers 0–1 (WMCR0, WMCR1),” for more information.

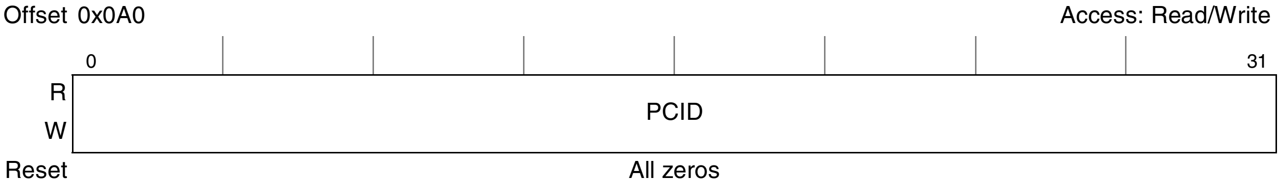


**Figure 25-17. Programmed Context ID Register (PCIDR)**

Table 25-23 describes the PCIDR field.

**Table 25-23. PCIDR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–31 | PCID | Programmed context ID. Contains the user-programmed context ID. Compared with current context ID for context-sensitive event triggering |

### 25.3.3.2 Current Context ID Register (CCIDR)

The current context ID register (CCIDR) shown in Figure 25-18 contains the current context ID. This register is written by software after a context switch and can be used to trigger events when compared with the programmed context ID register (PCIDR).

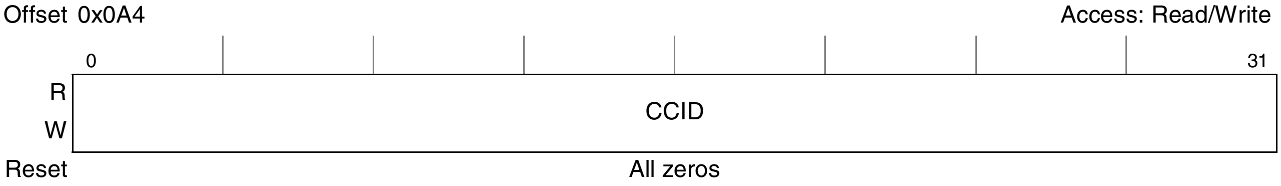


**Figure 25-18. Current Context ID Register (CCIDR)**

Table 25-24 describes the CCIDR field.

**Table 25-24. CCIDR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 0–31 | CCID | Current context ID. Set by user software. Typically loaded immediately following a context switch. Compared with user-programmed context ID for context-sensitive event triggering |

## 25.3.4 Trigger Out Function

TRIG_OUT provides a convenient mechanism for triggering external system monitors and diagnostic equipment such as logic analyzers. Note that READY is multiplexed with TRIG_OUT. See the last paragraph of Section 4.4.2, “Power-On Reset Sequence,” for more information about READY functionality.

When the trace buffer hit is selected by TOSR[SEL], TRIG_OUT is only meaningful if the trace buffer control register 0 (TBCR0) is properly configured to hit on a traceable event. The same holds true for the watchpoint monitor when the watchpoint monitor is selected by TOSR[SEL].

### 25.3.4.1 Trigger Out Source Register (TOSR)

The trigger out source register (TOSR) shown in Figure 25-19 specifies the source for TRIG_OUT. The three event-trigger sources are the following:

- The watchpoint monitor
- The trace buffer
- The performance monitor

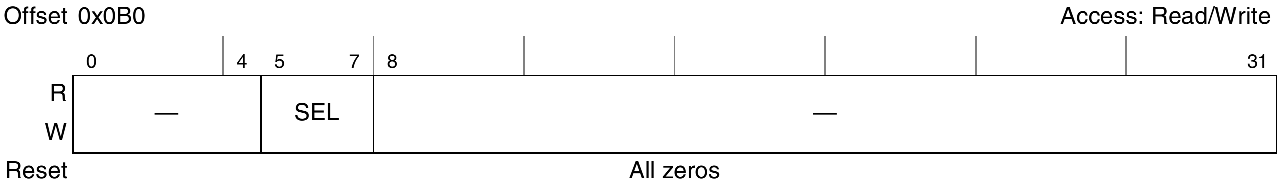


**Figure 25-19. Trigger Out Source Register (TOSR)**

Table 25-25 describes the TOSR fields.

Table 25-25. TOSR Field Descriptions

| Bits | Name | Description |
|---|---|---|
| 0–4 | — | Reserved |
| 5–7 | SEL | Select. Selects the source for TRIG_OUT<br>000 READY signal. Multiplexed with TRIG_OUT. Basic device state indicator. READY asserts whenever the device is not in reset or not asleep. See Chapter 4, “Reset, Clocking, and Initialization,” for more details about the reset sequence, and Chapter 23, “Global Utilities,” for more information about power management states.<br>001 Selects the watchpoint monitor hit indication<br>010 Selects the trace buffer hit indication<br>011 Selects the performance monitor overflow indication |
| 8–31 | — | Reserved |

## 25.4 Functional Description

The debug features on the MPC8536E use the LBC interfaces, and the DDR SDRAM interface.

### 25.4.1 Source and Target ID

Debug information that is common to all the interfaces is the source ID (SID). The transaction source ID provides enough information to determine which block or port originated a transaction including the distinction between instruction and data fetches from the processor core. Table 25-26 shows the values and interpretation for the 5-bit SID field. Note that the table also includes ports that are only slaves, such as local memory. These ports are always targets. As such, the value shown represents a target ID (TID) and not a source ID. For ports that can function in both capacities, the value indicates source ID when mastering transactions, and target ID when responding as slave. The TID field is only meaningful when one of the following participates in the transaction:

- The e500 coherency module (ECM) dispatch bus
- The watchpoint monitor (WMCR1[IFSEL] = 000)
- The trace buffer (TBCR1[IFSEL] = 000)

Table 25-26. Source and Target ID Values

| Value (Hex) | Source (or Target) Port | Value (Hex) | Source (or Target) Port |
|---|---|---|---|
| 00 | PCI | 10 | Local processor (instruction fetch) |
| 01 | PCI Express 2 | 11 | Local processor (data fetch) |
| 02 | PCI Express 1 | 12 | Reserved |
| 03 | PCI Express 3 | 13 | Reserved |
| 04 | Enhanced local bus controller | 14 | USB2 |
| 05 | USB1 | 15 | DMA |

Table 25-26. Source and Target ID Values (continued)

| Value (Hex) | Source (or Target) Port | Value (Hex) | Source (or Target) Port |
|---|---|---|---|
| 06 | Reserved | 16 | Reserved |
| 07 | Security | 17 | System access port (SAP) |
| 08 | SATA2/Configuration space | 18 | eTSEC1 |
| 09 | USB3 | 19 | Reserved |
| 0A | Boot sequencer | 1A | eTSEC3 |
| 0B | eSDHC | 1B | Reserved |
| 0C | Reserved | 1C | Reserved |
| 0D | SATA1 | 1D | Reserved |
| 0E | Reserved | 1E | Reserved |
| 0F | Local space (DDR) | 1F | Non-valid port indicator (reserved for debug info) |

## 25.4.2 DDR SDRAM Interface Debug

The DDR interface has two debug modes distinguished by which pins drive the debug information. In one mode, debug information (source ID, data valid) is multiplexed onto the ECC pins; the other mode uses the debug pins.

### 25.4.2.1 Debug Information on Debug Pins

If MSRCID0 is high when sampled during POR, the debug information from the DDR SDRAM interface is driven on MSRCID[0:4] and MDVAL. This POR value is captured in PORDBGMSR[MEM_SEL] as described in Section 23.4.1.5, “POR Debug Mode Status Register (PORDBGMSR).” In this mode, the source ID appears on MSRCID[0:4] during a RAS or CAS cycle. During any other cycle, the value of MSRCID[0:4] is all ones, which indicates idle cycles on the address/command interface. Similarly, MDVAL is asserted during valid data cycles on the DDR interface.

### 25.4.2.2 Debug Information on ECC Pins

If MSRCID1 is low when sampled during POR, debug information from the DDR SDRAM interface is selected to appear on MECC[0:5] as shown in Figure 25-1. In this mode, the ID value of the source port, (the source ID), appears on MECC[0:4] during a RAS or CAS cycle. During any other cycle the value of MECC[0:4] is all ones. A data-valid signal (DVAL) is driven on MECC5 during valid DDR SDRAM data cycles.

**NOTE**

In this mode, MECC[0:5] must be disconnected from all SDRAM devices to prevent contention on those lines.

### 25.4.3 Local Bus Interface Debug

If MSRCID0 is low when sampled during POR, the LBC is selected as the source for the debug information appearing on MSRCID[0:4] and MDVAL. For more information on this mode, see Section 13.1.3.2, “Source ID Debug Mode.”

### 25.4.4 Watchpoint Monitor

The watchpoint monitor (WM) can be programmed to arm and trigger on many different events including any of the following:

- External event (through TRIG_IN)
- A trace buffer event
- A performance monitor overflow event
- A comparison of the current and programmed context ID registers

A watchpoint event can be used in the following ways:

- Trigger a logic analyzer (using TRIG_OUT)
- Arm or trigger the trace buffer
- Trigger a performance monitor event

The large counters available in the performance monitor block and the interlock between it and the watchpoint monitor support sophisticated debug scenarios.

A WM trigger event may be composed of several events programmed in the watchpoint monitor control registers (WMCR0–WMCR1). Because the watchpoint monitor is disabled by default during POR, these registers must be initialized to make use of this debug feature. Note that the WM address mask register (WMAMR) and the type mask register (WMTMR) are cleared during POR. This means that the watchpoint monitor’s default behavior following a power-on reset is to trigger on any address and no transaction type. The reset value of WMCR0[TMD] is 0 which means transaction matching is enabled but since no transaction is selected (WMTMR=0), a match will never occur. Either the transaction matching must be disabled by setting WMCR0[TMD] to a value of 1, or valid transactions must be selected by setting one or more of the WMTMR bits to a value of 1.

#### 25.4.4.1 Watchpoint Monitor Performance Monitor Events

The WM can produce a performance monitor (PM) event with every trigger. This is accomplished by configuring the performance monitor to count WM events. For more information on this configuration see the events named ‘Number of watchpoint monitor hits’ and ‘Number of trace buffer hits’ in Table 24-10.

Multi-level triggers can be created using the watchpoint monitor, the performance monitor, and the trace buffer combined. For example, the WM can be programmed to trigger on events that also increment a PM counter (the performance monitor must also be programmed to respond to this event), the output of which (perfmon_overflow) could trigger the start of tracing in the trace buffer.

### 25.4.5 Trace Buffer

The trace buffer is a 256 × 64 array that can capture information about the internal processing of transactions to selected interfaces. The trace buffer controls are a superset of those for the watchpoint monitor. Close inspection of the trace buffer control registers (TBCR*n*) and the WM control registers (WMCR*n*) shows that trace buffer controls not needed for the WM are marked reserved in WMCR*n*. This permits using the trace buffer as a second watchpoint monitor by simply ignoring the trace options.

The trace buffer provides great flexibility about when to start tracing, when to stop tracing, and what to trace. The trace mode field, TBCR0[MODE], indicates when to trace: on every valid cycle, on a watchpoint monitor event, or when all the programmed events in the TBCR are met. This permits a user to program the trace condition in the watchpoint monitor and to program a start or stop condition in the trace buffer control register. The user can also program the TBCR with the conditions in which to stop tracing: on an event, or when the buffer is full. TBCR0[IFSEL] specifies which interface transactions are being captured.

The trace buffer can be programmed to trace the dispatch bus from any of the following:

- e500 coherency module (ECM)
- Outbound host interface to the PCI controller
- Host interface to the DDR controller

Transactions come into the ECM, arbitrate for common resources, and get dispatched to the target port. Information such as transaction types, source ID, and other attributes can be captured in any of the selected interfaces.

#### 25.4.5.1 Traced Data Formats (as a Function of TBCR1[IFSEL])

Figure 25-20 shows the trace buffer entry format for an ECM dispatch (CMD) transaction that is specified when TBCR1[IFSEL] = 000.

| | 0 4 | 5 9 | 10 13 | 14 18 | 19 31 | 32 63 |
|---|---|---|---|---|---|---|
| R<br>W | CMDTT | CMDSID | CMDTID | CMDBC | — | CMDADDR |
| Reset | All zeros | | | | | |

**Figure 25-20. e500 Coherency Module Dispatch (CMD) Trace Buffer Entry**

Table 25-27 describes the fields of CMD trace buffer entries.

**Table 25-27. CMD Trace Buffer Entry Field Descriptions (TBCR1[IFSEL] = 000)**

| Bits | Name | Function |
|---|---|---|
| 0–4 | CMDTT | Transaction type. Specifies the transaction type as shown in Table 25-12. For example, a value of zero indicates a write with local processor snoop condition. |
| 5–9 | CMDSID | Source ID. Identifies the source of the transaction as shown in Table 25-26. For example, a value of 010101 indicates that DMA is the transaction source. |
| 10–13 | CMDTID | Target ID. Identifies the target of the transaction as shown in Table 25-26. For example, a value of 010101 indicates that DMA is the transaction target. |

Table 25-27. CMD Trace Buffer Entry Field Descriptions (TBCR1[IFSEL] = 000) (continued)

| Bits | Name | Function |
|---|---|---|
| 14–18 | CMDBC | Byte count. Range: 32 to 1 where a value of 0 indicates 32 bytes.<br>00000 = 32 bytes<br>00001 = 1 byte<br>00010 = 2 bytes<br>. . .<br>11110 = 30 bytes<br>11111 = 31 bytes |
| 19–31 | — | Reserved |
| 32–63 | CMDADDR | Address bits 0–31 |

Figure 25-21 shows the trace buffer entry format for the DDR SDRAM interface, TBCR1[IFSEL] = 001.

| | 0 4 | 5 9 | 10 13 | 14 18 | 19 31 | 32 63 |
|---|---|---|---|---|---|---|
| R<br>W | DDRTT | DDRSID | — | DDRBC | — | DDRADDR |
| Reset | All zeros | | | | | |

Figure 25-21. DDR Trace Buffer Entry

Table 25-28 describes the fields of DDR SDRAM trace buffer entries when TBCR1[IFSEL] = 001.

Table 25-28. DDR Trace Buffer Entry Field Descriptions (TBCR1[IFSEL] = 001)

| Bits | Name | Function |
|---|---|---|
| 0–4 | DDRTT | Transaction type. Specifies the transaction type as shown in Table 25-12. For example, a value of all zeros maps to write. |
| 5–9 | DDRSID | Source ID. Specifies the source of the transaction as shown in Table 25-26. For example, a value of 010101 indicates that DMA is the transaction source, and so on. |
| 10–13 | — | Reserved |
| 14–18 | DDRBC | Byte count |
| 19–31 | — | Reserved |
| 32–63 | DDRADDR | Address bits 0–31 |

Figure 25-22 shows the PCI trace buffer entry format when TBCR1[IFSEL] = 010 or 101.

| | 0 4 | 5 9 | 10 11 | 12 31 | 32 63 |
|---|---|---|---|---|---|
| R<br>W | PCITT | PCISID | PCIBC | — | PCIADDR |
| Reset | All zeros | | | | |

Figure 25-22. PCI Trace Buffer Entry

**Table 25-29. PCI Trace Buffer Entry Field Descriptions (TBCR1[IFSEL] = 010)**

| Bits | Name | Function |
|---|---|---|
| 0–4 | PCITT | Transaction type. Specifies the transaction type as shown in Table 25-12. For example, a value of all zeros maps to write. |
| 5–9 | PCISID | Source ID. Identifies the source of the transaction as shown in Table 25-26. For example, a value of 010101 identifies DMA as the transaction source. |
| 10–11 | PCIBC | Byte count. The size of the transaction.<br>00 32 bytes<br>01 8 bytes<br>10 16 bytes<br>11 24 bytes |
| 12–31 | — | Reserved |
| 32–63 | PCIADDR | Address bits 0–31 |

Figure 25-23 shows the PCI Express trace buffer entry format when TBCR1[IFSEL] = 100 or 101 or 110.

| | 0 4 | 5 9 | 10 14 | 15 33 | 34 63 |
|---|---|---|---|---|---|
| R<br>W | PEXTT | PEXSID | PEXBC | — | PEXADDR |
| Reset | All zeros | | | | |

**Figure 25-23. PCI Express Trace Buffer Entry**

Table 25-30 describes the fields of PCI Express trace buffer entries when TBCR1[IFSEL] = 100 or 101 or 110.

**Table 25-30. PCI Express Trace Buffer Entry Field Descriptions (TBCR1[IFSEL] = 100 or 101 or 110)**

| Bits | Name | Function |
|---|---|---|
| 0–4 | PEXTT | Transaction type. Specifies the transaction type as shown in Table 25-12. For example, a value of all zeros maps to write. |
| 5–9 | PEXSID | Source ID. Identifies the source of the transaction as shown in Table 25-26. For example, a value of 010101 identifies DMA as the transaction source. For responses, this corresponds to Requestor's ID's bus number bits 3–7. |
| 10–14 | PCIBC | Byte count. The size of the transaction.<br>00000 4 bytes<br>00001 8 bytes<br>00010 12 bytes<br>...<br>11111 256 bytes |
| 15–33 | — | Reserved |
| 34–63 | PEXADDR | Address bits 31–2 |

## 25.5 Initialization

Configuring the appropriate control register must be the last step in the initialization sequence for either the watchpoint or trace buffer. That is, all required registers except the corresponding control register must be configured before any control register bits that enable watchpoint or trace events are set.

# Appendix A
# Complete List of Configuration, Control, and Status Registers

## A.1 General Utilities

The general utilities registers are the functional block-specific registers that occupy the first 256 Kbytes of CCSR space (0x0_0000–0x3_FFFF). Each functional block is allocated a 4-Kbyte address range for its registers within the general utilities space.

### A.1.1 Local Configuration Control

**Table A-1. Local Configuration Control Registers**

| Local Configuration Control—Block Base Address 0x0_0000 | | | | |
|---|---|---|---|---|
| **Offset** | **Register** | **Access** | **Reset** | **Section/Page** |
| 0x000 | CCSRBAR—Configuration, control, and status registers base address register | R/W | 0x000F_F700 | 4.3.1.1.2/4-5 |
| 0x008 | ALTCBAR—Alternate configuration base address register | R/W | 0x0000_0000 | 4.3.1.2.1/4-6 |
| 0x010 | ALTCAR—Alternate configuration attribute register | R/W | 0x0000_0000 | 4.3.1.2.2/4-6 |
| 0x020 | BPTR—Boot page translation register | R/W | 0x0000_0000 | 4.3.1.3.1/4-7 |

### A.1.2 Local Access Windows

**Table A-2. Local Access Window Registers**

| Local Access Windows—Block Base Address 0x0_0000 | | | | |
|---|---|---|---|---|
| **Offset** | **Register** | **Access** | **Reset** | **Section/Page** |
| 0xC08 | LAWBAR0—Local access window 0 base address register | R/W | 0x0000_0000 | 2.2.3.4/2-7 |
| 0xC10 | LAWAR0—Local access window 0 attribute register | R/W | 0x0000_0000 | 2.2.3.5/2-7 |
| 0xC28 | LAWBAR1—Local access window 1 base address register | R/W | 0x0000_0000 | 2.2.3.4/2-7 |
| 0xC30 | LAWAR1—Local access window 1 attribute register | R/W | 0x0000_0000 | 2.2.3.5/2-7 |
| 0xC48 | LAWBAR2—Local access window 2 base address register | R/W | 0x0000_0000 | 2.2.3.4/2-7 |
| 0xC50 | LAWAR2—Local access window 2 attribute register | R/W | 0x0000_0000 | 2.2.3.5/2-7 |
| 0xC68 | LAWBAR3—Local access window 3 base address register | R/W | 0x0000_0000 | 2.2.3.4/2-7 |
| 0xC70 | LAWAR3—Local access window 3 attribute register | R/W | 0x0000_0000 | 2.2.3.5/2-7 |

Table A-2. Local Access Window Registers (continued)

| Local Access Windows—Block Base Address 0x0_0000 | | | | |
|---|---|---|---|---|
| Offset | Register | Access | Reset | Section/Page |
| 0xC88 | LAWBAR4—Local access window 4 base address register | R/W | 0x0000_0000 | 2.2.3.4/2-7 |
| 0xC90 | LAWAR4—Local access window 4 attribute register | R/W | 0x0000_0000 | 2.2.3.5/2-7 |
| 0xCA8 | LAWBAR5—Local access window 5 base address register | R/W | 0x0000_0000 | 2.2.3.4/2-7 |
| 0xCB0 | LAWAR5—Local access window 5 attribute register | R/W | 0x0000_0000 | 2.2.3.5/2-7 |
| 0xCC8 | LAWBAR6—Local access window 6 base address register | R/W | 0x0000_0000 | 2.2.3.4/2-7 |
| 0xCD0 | LAWAR6—Local access window 6 attribute register | R/W | 0x0000_0000 | 2.2.3.5/2-7 |
| 0xCE8 | LAWBAR7—Local access window 7 base address register | R/W | 0x0000_0000 | 2.2.3.4/2-7 |
| 0xCF0 | LAWAR7—Local access window 7 attribute register | R/W | 0x0000_0000 | 2.2.3.5/2-7 |
| 0xD08 | LAWBAR8—Local access window 8 base address register | R/W | 0x0000_0000 | 2.2.3.4/2-7 |
| 0xD10 | LAWAR8—Local access window 8 attribute register | R/W | 0x0000_0000 | 2.2.3.5/2-7 |
| 0xD28 | LAWBAR9—Local access window 9 base address register | R/W | 0x0000_0000 | 2.2.3.4/2-7 |
| 0xD30 | LAWAR9—Local access window 9 attribute register | R/W | 0x0000_0000 | 2.2.3.5/2-7 |
| 0xD48 | LAWBAR10—Local access window 10 base address register | R/W | 0x0000_0000 | 2.2.3.4/2-7 |
| 0xD50 | LAWAR10—Local access window 10 attribute register | R/W | 0x0000_0000 | 2.2.3.5/2-7 |
| 0xD68 | LAWBAR11—Local access window 11 base address register | R/W | 0x0000_0000 | 2.2.3.4/2-7 |
| 0xD70 | LAWAR11—Local access window 11 attribute register | R/W | 0x0000_0000 | 2.2.3.5/2-7 |

## A.1.3 e500 Coherency Module (ECM)

Table A-3. ECM Registers

| ECM—Block Base Address 0x0_1000 | | | | |
|---|---|---|---|---|
| Offset | Register | Access | Reset | Section/Page |
| 0x000 | EEBACR—ECM CCB address configuration register | R/W | 0x0000_0003 | 7.2.1.1/7-3 |
| 0x010 | EEBPCR—ECM CCB port configuration register | R/W | 0x0*n*00_0000 | 7.2.1.2/7-4 |
| 0xBF8 | ECM IP Block Revision Register 1 | R | 0x0001_0000 | 7.2.1.3/7-5 |
| 0xBFC | ECM IP Block Revision Register 2 | R | 0x0000_0000 | 7.2.1.4/7-5 |
| 0xE00 | EEDR—ECM error detect register | w1c | 0x0000_0000 | 7.2.1.5/7-6 |
| 0xE08 | EEER—ECM error enable register | R/W | 0x0000_0000 | 7.2.1.6/7-7 |
| 0xE0C | EEATR—ECM error attributes capture register | R | 0x0000_0000 | 7.2.1.7/7-7 |
| 0xE10 | EELADR—ECM error low address capture register | R | 0x0000_0000 | 7.2.1.8/7-8 |
| 0xE14 | EEHADR—ECM error high address capture register | R | 0x0000_0000 | 7.2.1.9/7-9 |

### A.1.4 DDR Memory Controller

Table A-4. DDR Memory Controller Registers

| DDR Memory Controller—Block Base Address 0x0_2000 | | | | |
|---|---|---|---|---|
| Offset | Register | Access | Reset | Section/Page |
| 0x000 | CS0_BNDS—Chip select 0 memory bounds | R/W | 0x0000_0000 | 8.4.1.1/8-12 |
| 0x008 | CS1_BNDS—Chip select 1 memory bounds | R/W | 0x0000_0000 | 8.4.1.1/8-12 |
| 0x010 | CS2_BNDS—Chip select 2 memory bounds | R/W | 0x0000_0000 | 8.4.1.1/8-12 |
| 0x018 | CS3_BNDS—Chip select 3 memory bounds | R/W | 0x0000_0000 | 8.4.1.1/8-12 |
| 0x080 | CS0_CONFIG—Chip select 0 configuration | R/W | 0x0000_0000 | 8.4.1.2/8-13 |
| 0x084 | CS1_CONFIG—Chip select 1 configuration | R/W | 0x0000_0000 | 8.4.1.2/8-13 |
| 0x088 | CS2_CONFIG—Chip select 2 configuration | R/W | 0x0000_0000 | 8.4.1.2/8-13 |
| 0x08C | CS3_CONFIG—Chip select 3 configuration | R/W | 0x0000_0000 | 8.4.1.2/8-13 |
| 0x0C0 | CS0_CONFIG_2—Chip select 0 configuration 2 | R/W | 0x0000_0000 | 8.4.1.3/8-15 |
| 0x0C4 | CS1_CONFIG_2—Chip select 1 configuration 2 | R/W | 0x0000_0000 | 8.4.1.3/8-15 |
| 0x0C8 | CS2_CONFIG_2—Chip select 2 configuration 2 | R/W | 0x0000_0000 | 8.4.1.3/8-15 |
| 0x0CC | CS3_CONFIG_2—Chip select 3 configuration 2 | R/W | 0x0000_0000 | 8.4.1.3/8-15 |
| 0x100 | TIMING_CFG_3—DDR SDRAM timing configuration 3 | R/W | 0x0000_0000 | 8.4.1.4/8-16 |
| 0x104 | TIMING_CFG_0—DDR SDRAM timing configuration 0 | R/W | 0x0011_0105 | 8.4.1.5/8-17 |
| 0x108 | TIMING_CFG_1—DDR SDRAM timing configuration 1 | R/W | 0x0000_0000 | 8.4.1.6/8-19 |
| 0x10C | TIMING_CFG_2—DDR SDRAM timing configuration 2 | R/W | 0x0000_0000 | 8.4.1.7/8-21 |
| 0x110 | DDR_SDRAM_CFG—DDR SDRAM control configuration | R/W | 0x0200_0000 | 8.4.1.8/8-23 |
| 0x114 | DDR_SDRAM_CFG_2—DDR SDRAM control configuration 2 | R/W | 0x0000_0000 | 8.4.1.9/8-26 |
| 0x118 | DDR_SDRAM_MODE—DDR SDRAM mode configuration | R/W | 0x0000_0000 | 8.4.1.10/8-29 |
| 0x11C | DDR_SDRAM_MODE_2—DDR SDRAM mode configuration 2 | R/W | 0x0000_0000 | 8.4.1.11/8-29 |
| 0x120 | DDR_SDRAM_MD_CNTL—DDR SDRAM mode control | R/W | 0x0000_0000 | 8.4.1.12/8-30 |
| 0x124 | DDR_SDRAM_INTERVAL—DDR SDRAM interval configuration | R/W | 0x0000_0000 | 8.4.1.13/8-33 |
| 0x128 | DDR_DATA_INIT—DDR SDRAM data initialization | R/W | 0x0000_0000 | 8.4.1.14/8-33 |
| 0x130 | DDR_SDRAM_CLK_CNTL—DDR SDRAM clock control | R/W | 0x0200_0000 | 8.4.1.15/8-34 |
| 0x140–0x144 | Reserved | — | — | — |
| 0x148 | DDR_INIT_ADDR—DDR training initialization address | R/W | 0x0000_0000 | 8.4.1.16/8-34 |
| 0x14C | DDR_INIT_EXT_ADDRESS—DDR training initialization extended address | R/W | 0x0000_0000 | 8.4.1.17/8-35 |
| 0x150–0x15F | Reserved | — | — | — |
| 0x160 | TIMING_CFG_4— DDR SDRAM timing configuration 4 | R/W | 0x0000_0000 | 8.4.1.18/8-36 |
| 0x164 | TIMING_CFG_5— DDR SDRAM timing configuration 5 | R/W | 0x0000_0000 | 8.4.1.19/8-37 |

## Table A-4. DDR Memory Controller Registers (continued)

| DDR Memory Controller—Block Base Address 0x0_2000 | | | | |
|---|---|---|---|---|
| **Offset** | **Register** | **Access** | **Reset** | **Section/Page** |
| 0x168–0x16F | Reserved | — | — | — |
| 0x170 | DDR_ZQ_CNTL— DDR ZQ calibration control | R/W | 0x0000_0000 | 8.4.1.20/8-39 |
| 0x174 | DDR_WRLVL_CNTL— DDR write leveling control | R/W | 0x0000_0000 | 8.4.1.21/8-40 |
| 0x178 | Reserved | — | — | — |
| 0x17C | DDR_SR_CNTR — DDR Self Refresh Coutner | R/W | 0x0000_0000 | 8.4.1.22/8-43 |
| 0x180 | DDR_SDRAM_RCW_1 — DDR Register Control Words 1 | R/W | 0x0000_0000 | 8.4.1.23/8-44 |
| 0x184 | DDR_SDRAM_RCW_2 — DDR Register Control Words 2 | R/W | 0x0000_0000 | 8.4.1.24/8-45 |
| 0x188–0xB1F | Reserved | — | — | — |
| 0xB20 | DDRDSR_1—DDR Debug Status Register 1 | R | 0x0000_0000 | 8.4.1.25/8-46 |
| 0xB24 | DDRDSR_2—DDR Debug Status Register 2 | R | 0x0000_0000 | 8.4.1.26/8-47 |
| 0xB28 | DDRCDR_1—DDR Control Driver Register 1 | R/W | 0x0000_0000 | 8.4.1.27/8-47 |
| 0xB2C | DDRCDR_2—DDR Control Driver Register 2 | R/W | 0x0000_0000 | 8.4.1.28/8-50 |
| 0xB30–0xBF7 | Reserved | — | — | — |
| 0xBF8 | DDR_IP_REV1—DDR IP block revision 1 | R | 0x*nnnn_nnnn*[1] | 8.4.1.29/8-50 |
| 0xBFC | DDR_IP_REV2—DDR IP block revision 2 | R | 0x00*nn*_00*nn*[1] | 8.4.1.30/8-51 |
| 0xE00 | DATA_ERR_INJECT_HI—Memory data path error injection mask high | R/W | 0x0000_0000 | 8.4.1.31/8-51 |
| 0xE04 | DATA_ERR_INJECT_LO—Memory data path error injection mask low | R/W | 0x0000_0000 | 8.4.1.32/8-52 |
| 0xE08 | ECC_ERR_INJECT—Memory data path error injection mask ECC | R/W | 0x0000_0000 | 8.4.1.33/8-52 |
| 0xE20 | CAPTURE_DATA_HI—Memory data path read capture high | R/W | 0x0000_0000 | 8.4.1.34/8-53 |
| 0xE24 | CAPTURE_DATA_LO—Memory data path read capture low | R/W | 0x0000_0000 | 8.4.1.35/8-54 |
| 0xE28 | CAPTURE_ECC—Memory data path read capture ECC | R/W | 0x0000_0000 | 8.4.1.36/8-54 |
| 0xE40 | ERR_DETECT—Memory error detect | w1c | 0x0000_0000 | 8.4.1.37/8-54 |
| 0xE44 | ERR_DISABLE—Memory error disable | R/W | 0x0000_0000 | 8.4.1.38/8-56 |
| 0xE48 | ERR_INT_EN—Memory error interrupt enable | R/W | 0x0000_0000 | 8.4.1.39/8-57 |
| 0xE4C | CAPTURE_ATTRIBUTES—Memory error attributes capture | R/W | 0x0000_0000 | 8.4.1.40/8-58 |
| 0xE50 | CAPTURE_ADDRESS—Memory error address capture | R/W | 0x0000_0000 | 8.4.1.41/8-58 |
| 0xE54 | CAPTURE_EXT_ADDRESS—Memory error extended address capture | R/W | 0x0000_0000 | 8.4.1.42/8-59 |
| 0xE58 | ERR_SBE—Single-Bit ECC memory error management | R/W | 0x0000_0000 | 8.4.1.43/8-59 |

[1] Implementation-dependent reset values are listed in specified section/page.

### A.1.5 I$^2$C Controllers

Table A-5. I$^2$C Controller 1 & 2 Registers

| I$^2$C Controller 1—Block Base Address 0x0_3000<br>I$^2$C Controller 2—Block Base Address 0x0_3100 | | | | |
|---|---|---|---|---|
| Offset | Register | Access | Reset | Section/Page |
| I$^2$C1 Registers | | | | |
| 0x000 | I2CADR—I$^2$C address register | R/W | 0x00 | 11.3.1.1/11-6 |
| 0x004 | I2CFDR—I$^2$C frequency divider register | R/W | 0x00 | 11.3.1.2/11-6 |
| 0x008 | I2CCR—I$^2$C control register | Mixed | 0x00 | 11.3.1.3/11-7 |
| 0x00C | I2CSR—I$^2$C status register | Mixed | 0x81 | 11.3.1.4/11-9 |
| 0x010 | I2CDR—I$^2$C data register | R/W | 0x00 | 11.3.1.5/11-10 |
| 0x014 | I2CDFSRR—I$^2$C digital filter sampling rate register | R/W | 0x10 | 11.3.1.6/11-11 |
| I$^2$C2 Registers | | | | |
| 0x100–<br>0x114 | I$^2$C2 Registers[1] | | | |

[1] I$^2$C2 has the same memory-mapped registers that are described for I$^2$C1 from 0x000 to 0x014, except the offsets range from 0x100 to 0x114.

### A.1.6 DUART

Table A-6. DUARTRegisters

| DUART1 & DUART2—Block Base Address 0x0_4000 | | | | |
|---|---|---|---|---|
| Offset | Register | Access | Reset | Section/Page |
| UART0 Registers | | | | |
| 0x500 | URBR—ULCR[DLAB] = 0 UART0 receiver buffer register | R | 0x00 | 12.3.1.1/12-5 |
| 0x500 | UTHR—ULCR[DLAB] = 0 UART0 transmitter holding register | W | 0x00 | 12.3.1.2/12-5 |
| 0x500 | UDLB—ULCR[DLAB] = 1 UART0 divisor least significant byte register | R/W | 0x00 | 12.3.1.3/12-6 |
| 0x501 | UIER—ULCR[DLAB] = 0 UART0 interrupt enable register | R/W | 0x00 | 12.3.1.4/12-7 |
| 0x501 | UDMB—ULCR[DLAB] = 1 UART0 divisor most significant byte register | R/W | 0x00 | 12.3.1.3/12-6 |
| 0x502 | UIIR—ULCR[DLAB] = 0 UART0 interrupt ID register | R | 0x01 | 12.3.1.5/12-8 |
| 0x502 | UFCR—ULCR[DLAB] = 0 UART0 FIFO control register | W | 0x00 | 12.3.1.6/12-10 |
| 0x502 | UAFR—ULCR[DLAB] = 1 UART0 alternate function register | R/W | 0x00 | 12.3.1.7/12-11 |
| 0x503 | ULCR—ULCR[DLAB] = x UART0 line control register | R/W | 0x00 | 12.3.1.8/12-11 |
| 0x504 | UMCR—ULCR[DLAB] = x UART0 modem control register | R/W | 0x00 | 12.3.1.9/12-14 |
| 0x505 | ULSR—ULCR[DLAB] = x UART0 line status register | R | 0x60 | 12.3.1.10/12-15 |
| 0x506 | UMSR—ULCR[DLAB] = x UART0 modem status register | R | 0x00 | 12.3.1.11/12-16 |

Table A-6. DUARTRegisters (continued)

| DUART1 & DUART2—Block Base Address 0x0_4000 | | | | |
|---|---|---|---|---|
| **Offset** | **Register** | **Access** | **Reset** | **Section/Page** |
| 0x507 | USCR—ULCR[DLAB] = x UART0 scratch register | R/W | 0x00 | 12.3.1.12/12-17 |
| 0x510 | UDSR—ULCR[DLAB] = x UART0 DMA status register | R | 0x01 | 12.3.1.13/12-17 |
| **UART1 Registers** | | | | |
| 0x600–<br>0x610 | UART1 Registers[1] | | | |

[1] UART1 has the same memory-mapped registers that are described for UART0 from 0x500 to 0x510, except the offsets range from 0x600 to 0x610.

## A.1.7 Enhanced Local Bus Controller

Table A-7. Enhanced Local Bus Controller Registers

| Enhanced Local Bus Controller—Block Base Address 0x0_5000 | | | | |
|---|---|---|---|---|
| **Offset** | **Register** | **Access** | **Reset** | **Section/Page** |
| 0x000 | BR0—Base register 0 | R/W | 0x0000_*nnnn* | 13.3.1.1/13-11 |
| 0x008 | BR1—Base register 1 | R/W | 0x0000_0000 | 13.3.1.1/13-11 |
| 0x010 | BR2—Base register 2 | R/W | 0x0000_0000 | 13.3.1.1/13-11 |
| 0x018 | BR3—Base register 3 | R/W | 0x0000_0000 | 13.3.1.1/13-11 |
| 0x020 | BR4—Base register 4 | R/W | 0x0000_0000 | 13.3.1.1/13-11 |
| 0x028 | BR5—Base register 5 | R/W | 0x0000_0000 | 13.3.1.1/13-11 |
| 0x030 | BR6—Base register 6 | R/W | 0x0000_0000 | 13.3.1.1/13-11 |
| 0x038 | BR7—Base register 7 | R/W | 0x0000_0000 | 13.3.1.1/13-11 |
| 0x004 | OR0—Options register 0 | R/W | 0x0000_0FF7 | 13.3.1.2/13-12 |
| 0x00C | OR1—Options register 1 | R/W | 0x0000_0000 | 13.3.1.2/13-12 |
| 0x014 | OR2—Options register 2 | R/W | 0x0000_0000 | 13.3.1.2/13-12 |
| 0x01C | OR3—Options register 3 | R/W | 0x0000_0000 | 13.3.1.2/13-12 |
| 0x024 | OR4—Options register 4 | R/W | 0x0000_0000 | 13.3.1.2/13-12 |
| 0x02C | OR5—Options register 5 | R/W | 0x0000_0000 | 13.3.1.2/13-12 |
| 0x034 | OR6—Options register 6 | R/W | 0x0000_0000 | 13.3.1.2/13-12 |
| 0x03C | OR7—Options register 7 | R/W | 0x0000_0000 | 13.3.1.2/13-12 |
| 0x040–<br>0x064 | Reserved | — | — | — |
| 0x068 | MAR—UPM address register | R/W | 0x0000_0000 | 13.3.1.3/13-20 |
| 0x06C | Reserved | — | — | — |

**Table A-7. Enhanced Local Bus Controller Registers (continued)**

| Enhanced Local Bus Controller—Block Base Address 0x0_5000 | | | | |
|---|---|---|---|---|
| **Offset** | **Register** | **Access** | **Reset** | **Section/Page** |
| 0x070 | MAMR—UPMA mode register | R/W | 0x0000_0000 | 13.3.1.4/13-21 |
| 0x074 | MBMR—UPMB mode register | R/W | 0x0000_0000 | 13.3.1.4/13-21 |
| 0x078 | MCMR—UPMC mode register | R/W | 0x0000_0000 | 13.3.1.4/13-21 |
| 0x07C–0x080 | Reserved | — | — | — |
| 0x084 | MRTPR—Memory refresh timer prescaler register | R/W | 0x0000_0000 | 13.3.1.5/13-23 |
| 0x088 | MDR—UPM/FCM data register | R/W | 0x0000_0000 | 13.3.1.6/13-23 |
| 0x08C | Reserved | — | — | — |
| 0x090 | LSOR—Special operation initiation register | R/W | 0x0000_0000 | 13.3.1.7/13-24 |
| 0x094–0x09C | Reserved | — | — | — |
| 0x0A0 | LURT—UPM refresh timer | R/W | 0x0000_0000 | 13.3.1.4/13-21 |
| 0x0A4–0x0AC | Reserved | — | — | — |
| 0x0B0 | LTESR—Transfer error status register | w1c | 0x0000_0000 | 13.3.1.9/13-26 |
| 0x0B4 | LTEDR—Transfer error disable register | R/W | 0x0000_0000 | 13.3.1.10/13-28 |
| 0x0B8 | LTEIR—Transfer error interrupt register | R/W | 0x0000_0000 | 13.3.1.11/13-29 |
| 0x0BC | LTEATR—Transfer error attributes register | R/W | 0x0000_0000 | 13.3.1.12/13-30 |
| 0x0C0 | LTEAR—Transfer error address register | R/W | 0x0000_0000 | 13.3.1.13/13-31 |
| 0x0C4 | LTECCR—Transfer error ECC register | w1c | 0x0000_0000 | 13.3.1.14/13-31 |
| 0x0C8–0x0CC | Reserved | — | — | — |
| 0x0D0 | LBCR—Configuration register | R/W | | 13.3.1.15/13-32 |
| 0x0D4 | LCRR—Clock ratio register | R/W | 0x8000_000*n* | 13.3.1.16/13-34 |
| 0x0D8–0x0DC | Reserved | — | — | — |
| 0x0E0 | FMR—Flash mode register | R/W | 0x0000_0*n*00 | 13.3.1.17/13-35 |
| 0x0E4 | FIR—Flash instruction register | R/W | 0x0000_0000 | 13.3.1.18/13-37 |
| 0x0E8 | FCR—Flash command register | R/W | 0x0000_0000 | 13.3.1.19/13-38 |
| 0x0EC | FBAR—Flash block address register | R/W | 0x0000_0000 | 13.3.1.20/13-39 |
| 0x0F0 | FPAR—Flash page address register | R/W | 0x0000_0000 | 13.3.1.21/13-39 |
| 0x0F4 | FBCR—Flash byte count register | R/W | 0x0000_0000 | 13.3.1.22/13-41 |
| 0x0F8–0x0FC | Reserved | — | — | — |

Table A-7. Enhanced Local Bus Controller Registers (continued)

| Enhanced Local Bus Controller—Block Base Address 0x0_5000 | | | | |
|---|---|---|---|---|
| **Offset** | **Register** | **Access** | **Reset** | **Section/Page** |
| 0x100 | FECC0—Flash ECC block 0 register | R/O | 0x0000_0000 | 13.3.1.23/13-41 |
| 0x104 | FECC1—Flash ECC block 1 register | R/O | 0x0000_0000 | |
| 0x108 | FECC2—Flash ECC block 2 register | R/O | 0x0000_0000 | |
| 0x10C | FECC3—Flash ECC block 3 register | R/O | 0x0000_0000 | |

## A.1.8 Enhanced Serial Peripheral Interface (eSPI)

Table A-8. Serial Peripheral Interface Registers

| Serial Peripheral Interface—Block Base Address 0x0_7000 | | | | |
|---|---|---|---|---|
| **Offset** | **Register** | **Access** | **Reset** | **Section/Page** |
| 0x000 | SPMODE—eSPI mode register | R/W | 0x0000_100F | 18.3.1.1/18-6 |
| 0x004 | SPIE—eSPI event register | Mixed | 0x0020_0000 | 18.3.1.2/18-6 |
| 0x008 | SPIM—eSPI mask register | R/W | 0x0000_0000 | 18.3.1.3/18-7 |
| 0x00C | SPCOM—eSPI command register | W | 0x0000_0000 | 18.3.1.4/18-9 |
| 0x010 | SPITF—eSPI transmit FIFO access register | W | — | 18.3.1.5/18-10 |
| 0x014 | SPIRF—eSPI receive FIFO access register | R | — | 18.3.1.6/18-11 |
| 0x018–0x01C | Reserved | — | | |
| 0x020 | SPMODE0—eSPI CS0 mode register | R/W | 0x0010_0000 | 18.3.1.7/18-12 |
| 0x024 | SPMODE1—eSPI CS1 mode register | R/W | 0x0010_0000 | 18.3.1.7/18-12 |
| 0x028 | SPMODE2—eSPI CS2 mode register | R/W | 0x0010_0000 | 18.3.1.7/18-12 |
| 0x02C | SPMODE3—eSPI CS3 mode register | R/W | 0x0010_0000 | 18.3.1.7/18-12 |

## A.1.9 PCI Controller

Table A-9. PCI Controller Registers

| PCI Controller—Block Base Address 0x0_8000 | | | | |
|---|---|---|---|---|
| **Offset** | **Register** | **Access** | **Reset** | **Section/Page** |
| **PCI Configuration Access Registers** | | | | |
| 0x000 | CFG_ADDR—PCI configuration address | R/W | 0x0000_0000 | 16.3.1.1.1/16-14 |
| 0x004 | CFG_DATA—PCI configuration data | R/W | 0x0000_0000 | 16.3.1.1.2/16-15 |
| 0x008 | INT_ACK—PCI interrupt acknowledge | R | 0x0000_0000 | 16.3.1.1.3/16-15 |
| 0x00C–<br>0xBFC | Reserved | — | — | — |

Table A-9. PCI Controller Registers (continued)

| PCI Controller—Block Base Address 0x0_8000 | | | | |
|---|---|---|---|---|
| **Offset** | **Register** | **Access** | **Reset** | **Section/Page** |
| **PCI ATMU Registers—Outbound and Inbound** | | | | |
| 0xC00–0xC3C–Outbound Window 0 (default) | | | | |
| 0xC00 | POTAR0—PCI outbound window 0 (default) translation address register | R/W | 0x0000_0000 | 16.3.1.2.1/16-16 |
| 0xC04 | POTEAR0—PCI outbound window 0 (default) translation extended address register | R/W | 0x0000_0000 | 16.3.1.2.2/16-16 |
| 0xC08 | Reserved | — | — | |
| 0xC0C | Reserved | — | — | |
| 0xC10 | POWAR0—PCI outbound window 0 (default) attributes register | R/W | 0x8004_401F | 16.3.1.2.4/16-17 |
| 0xC14–0xC1C | Reserved | — | — | |
| 0xC20–0xC3C—Outbound Window 1 | | | | |
| 0xC20 | POTAR1—PCI outbound window 1 translation address register | R/W | 0x0000_0000 | 16.3.1.2.1/16-16 |
| 0xC24 | POTEAR1—PCI outbound window 1 translation extended address register | R/W | 0x0000_0000 | 16.3.1.2.2/16-16 |
| 0xC28 | POWBAR1—PCI outbound window 1 base address register | R/W | 0x0000_0000 | 16.3.1.2.3/16-17 |
| 0xC2C | Reserved | — | — | |
| 0xC30 | POWAR1—PCI outbound window 1 attributes register | R/W | 0x0000_0000 | 16.3.1.2.4/16-17 |
| 0xC34–0xC3C | Reserved | — | — | |
| 0xC40–0xC5C—Outbound Window 2 | | | | |
| 0xC40 | POTAR2—PCI outbound window 2 translation address register | R/W | 0x0000_0000 | 16.3.1.2.1/16-16 |
| 0xC44 | POTEAR2—PCI outbound window 2 translation extended address register | R/W | 0x0000_0000 | 16.3.1.2.2/16-16 |
| 0xC48 | POWBAR2—PCI outbound window 2 base address register | R/W | 0x0000_0000 | 16.3.1.2.3/16-17 |
| 0xC4C | Reserved | — | — | |
| 0xC50 | POWAR2—PCI outbound window 2 attributes register | R/W | 0x0000_0000 | 16.3.1.2.4/16-17 |
| 0xC54–0xC5C | Reserved | — | — | |
| 0xC60–0xC7C—Outbound Window 3 | | | | |
| 0xC60 | POTAR3—PCI outbound window 3 translation address register | R/W | 0x0000_0000 | 16.3.1.2.1/16-16 |
| 0xC64 | POTEAR3—PCI outbound window 3 translation extended address register | R/W | 0x0000_0000 | 16.3.1.2.2/16-16 |
| 0xC68 | POWBAR3—PCI outbound window 3 base address register | R/W | 0x0000_0000 | 16.3.1.2.3/16-17 |
| 0xC6C | Reserved | — | — | |

Table A-9. PCI Controller Registers (continued)

| PCI Controller—Block Base Address 0x0_8000 | | | | |
|---|---|---|---|---|
| **Offset** | **Register** | **Access** | **Reset** | **Section/Page** |
| 0xC70 | POWAR3—PCI outbound window 3 attributes register | R/W | 0x0000_0000 | 16.3.1.2.4/16-17 |
| 0xC74–0xC7C | Reserved | — | — | |
| 0xC80–0xC9C—Outbound Window 4 | | | | |
| 0xC80 | POTAR4—PCI outbound window 4 translation address register | R/W | 0x0000_0000 | 16.3.1.2.1/16-16 |
| 0xC84 | POTEAR4—PCI outbound window 4 translation extended address register | R/W | 0x0000_0000 | 16.3.1.2.2/16-16 |
| 0xC88 | POWBAR4—PCI outbound window 4 base address register | R/W | 0x0000_0000 | 16.3.1.2.3/16-17 |
| 0xC8C | Reserved | — | — | |
| 0xC90 | POWAR4—PCI outbound window 4 attributes register | R/W | 0x0000_0000 | 16.3.1.2.4/16-17 |
| 0xC94–0xD9C | Reserved | — | — | |
| 0xDA0–0xDBC–Inbound Window 3 | | | | |
| 0xDA0 | PITAR3—PCI inbound window 3 translation address register | R/W | 0x0000_0000 | 16.3.1.3.1/16-20 |
| 0xDA4 | Reserved | — | — | |
| 0xDA8 | PIWBAR3—PCI inbound window 3 base address register | R/W | 0x0000_0000 | 16.3.1.3.2/16-20 |
| 0xDAC | PIWBEAR3—PCI inbound window 3 base extended address register | R/W | 0x0000_0000 | 16.3.1.3.3/16-21 |
| 0xDB0 | PIWAR3—PCI inbound window 3 attributes register | R/W | 0x0000_0000 | 16.3.1.3.4/16-21 |
| 0xDB4–0xDBC | Reserved | — | — | |
| 0xDC0–0xDDC–Inbound Window 2 | | | | |
| 0xDC0 | PITAR2—PCI inbound window 2 translation address register | R/W | 0x0000_0000 | 16.3.1.3.1/16-20 |
| 0xDC4 | Reserved | — | — | |
| 0xDC8 | PIWBAR2—PCI inbound window 2 base address register | R/W | 0x0000_0000 | 16.3.1.3.2/16-20 |
| 0xDCC | PIWBEAR2—PCI inbound window 2 base extended address register | R/W | 0x0000_0000 | 16.3.1.3.3/16-21 |
| 0xDD0 | PIWAR2—PCI inbound window 2 attributes register | R/W | 0x0000_0000 | 16.3.1.3.4/16-21 |
| 0xDD4–0xDDC | Reserved | — | — | |
| 0xDE0–0xDFC–Inbound Window 1 | | | | |
| 0xDE0 | PITAR1—PCI inbound window 1 translation address register | R/W | 0x0000_0000 | 16.3.1.3.1/16-20 |
| 0xDE4 | Reserved | — | — | |
| 0xDE8 | PIWBAR1—PCI inbound window 1 base address register | R/W | 0x0000_0000 | 16.3.1.3.2/16-20 |
| 0xDEC | Reserved | — | — | |
| 0xDF0 | PIWAR1—PCI inbound window 1 attributes register | R/W | 0x0000_0000 | 16.3.1.3.4/16-21 |
| 0xDF4–0xDFC | Reserved | — | — | |

**Table A-9. PCI Controller Registers (continued)**

| PCI Controller—Block Base Address 0x0_8000 | | | | |
|---|---|---|---|---|
| **Offset** | **Register** | **Access** | **Reset** | **Section/Page** |
| **PCI Error Management Registers** | | | | |
| 0xE00 | ERR_DR—PCI error detect register | w1c | 0x0000_0000 | 16.3.1.4.1/16-24 |
| 0xE04 | ERR_CAP_DR—PCI error capture disabled register | R/W | 0x0000_0000 | 16.3.1.4.2/16-25 |
| 0xE08 | ERR_EN—PCI error enable register | R/W | 0x0000_0000 | 16.3.1.4.3/16-26 |
| 0xE0C | ERR_ATTRIB—PCI error attributes capture register | R/W | 0x0000_0000 | 16.3.1.4.4/16-27 |
| 0xE10 | ERR_ADDR—PCI error address capture register | R/W | 0x0000_0000 | 16.3.1.4.5/16-28 |
| 0xE14 | ERR_EXT_ADDR—PCI error extended address capture register | R/W | 0x0000_0000 | 16.3.1.4.6/16-28 |
| 0xE18 | ERR_DL—PCI error data low capture register | R/W | 0x0000_0000 | 16.3.1.4.7/16-29 |
| 0xE1C | ERR_DH—PCI error data high capture register | R/W | 0x0000_0000 | 16.3.1.4.8/16-29 |
| 0xE20 | GAS_TIMR—PCI gasket timer register | R/W | 0x0100_3FFF | 16.3.1.4.9/16-29 |
| 0xE28–<br>0xEFC | Reserved | — | — | |
| 0xF00–<br>0xFFC | Reserved for debug | — | — | |

## A.1.10 PCI Express Controllers

**Table A-10. PCI Express Controller 1 & 2 Registers**

| PCI Express Controller 1—Block Base Address 0x0_A000<br>PCI Express Controller 2—Block Base Address 0x0_9000<br>PCI Express Controller 3—Block Base Address 0x0_B000 | | | | |
|---|---|---|---|---|
| **Offset** | **Register** | **Access** | **Reset** | **Section/Page** |
| **PCI Express Configuration Access Registers** | | | | |
| 0x000 | PEX_CONFIG_ADDR—PCI Express configuration address register | R/W | 0x0000_0000 | 17.3.2.1/17-10 |
| 0x004 | PEX_CONFIG_DATA—PCI Express configuration data register | R/W | 0x0000_0000 | 17.3.2.2/17-10 |
| 0x008 | Reserved | — | — | |
| 0x00C | PEX_OTB_CPL_TOR—PCI Express outbound completion timeout register | R/W | 0x0010_FFFF | 17.3.2.3/17-11 |
| 0x010 | PEX_CONF_RTY_TOR—PCI Express configuration retry timeout register | R/W | 0x0400_FFFF | 17.3.2.4/17-12 |
| 0x014 | PEX_CONFIG—PCI Express configuration register | R/W | 0x0000_0000 | 17.3.2.5/17-12 |
| 0x018–<br>0x01C | Reserved | — | — | |
| **PCI Express Power Management Event & Message Registers** | | | | |
| 0x020 | PEX_PME_MES_DR—PCI Express PME & message detect register | w1c | 0x0000_0000 | 17.3.3.1/17-13 |

Table A-10. PCI Express Controller 1 & 2 Registers (continued)

| PCI Express Controller 1—Block Base Address 0x0_A000<br>PCI Express Controller 2—Block Base Address 0x0_9000<br>PCI Express Controller 3—Block Base Address 0x0_B000 | | | | |
|---|---|---|---|---|
| **Offset** | **Register** | **Access** | **Reset** | **Section/Page** |
| 0x024 | PEX_PME_MES_DISR—PCI Express PME & message disable register | R/W | 0x0000_0000 | 17.3.3.2/17-15 |
| 0x028 | PEX_PME_MES_IER—PCI Express PME & message interrupt enable register | R/W | 0x0000_0000 | 17.3.3.3/17-16 |
| 0x02C | PEX_PMCR—PCI Express power management command register | R/W | 0x0000_0000 | 17.3.3.4/17-18 |
| 0x030–0xBF4 | Reserved | — | — | |
| **PCI Express IP Block Revision Registers** | | | | |
| 0xBF8 | IP block revision register 1 (PEX_IP_BLK_REV1) | R | 0x0208_0100 | 17.3.4.1/17-19 |
| 0xBFC | IP block revision register 2 (PEX_IP_BLK_REV2) | R | 0x0000_0000 | 17.3.4.2/17-19 |
| **PCI Express ATMU Registers** | | | | |
| **Outbound Window 0 (Default)** | | | | |
| 0xC00 | PEXOTAR0—PCI Express outbound translation address register 0 (default) | R/W | 0x0000_0000 | 17.3.5.1.1/17-20 |
| 0xC04 | PEXOTEAR0—PCI Express outbound translation extended address register 0 (default) | R/W | 0x0000_0000 | 17.3.5.1.2/17-21 |
| 0xC08–0xC0C | Reserved | — | — | |
| 0xC10 | PEXOWAR0—PCI Express outbound window attributes register 0 (default) | Mixed | 0x8004_4023 | 17.3.5.1.4/17-22 |
| 0xC14–0xC1C | Reserved | — | — | |
| **Outbound Window 1** | | | | |
| 0xC20 | PEXOTAR1—PCI Express outbound translation address register 1 | R/W | 0x0000_0000 | 17.3.5.1.1/17-20 |
| 0xC24 | PEXOTEAR1—PCI Express outbound translation extended address register 1 | R/W | 0x0000_0000 | 17.3.5.1.2/17-21 |
| 0xC28 | PEXOWBAR1—PCI Express outbound window base address register 1 | R/W | 0x0000_0000 | 17.3.5.1.3/17-22 |
| 0xC2C | Reserved | — | — | |
| 0xC30 | PEXOWAR1—PCI Express outbound window attributes register 1 | R/W | 0x0004_4023 | 17.3.5.1.4/17-22 |
| 0xC34–0xC3C | Reserved | — | — | |
| **Outbound Window 2** | | | | |
| 0xC40 | PEXOTAR2—PCI Express outbound translation address register 2 | R/W | 0x0000_0000 | 17.3.5.1.1/17-20 |

Table A-10. PCI Express Controller 1 & 2 Registers (continued)

| PCI Express Controller 1—Block Base Address 0x0_A000<br>PCI Express Controller 2—Block Base Address 0x0_9000<br>PCI Express Controller 3—Block Base Address 0x0_B000 | | | | |
|---|---|---|---|---|
| **Offset** | **Register** | **Access** | **Reset** | **Section/Page** |
| 0xC44 | PEXOTEAR2—PCI Express outbound translation extended address register 2 | R/W | 0x0000_0000 | 17.3.5.1.2/17-21 |
| 0xC48 | PEXOWBAR2—PCI Express outbound window base address register 2 | R/W | 0x0000_0000 | 17.3.5.1.3/17-22 |
| 0xC4C | Reserved | — | — | |
| 0xC50 | PEXOWAR2—PCI Express outbound window attributes register 2 | R/W | 0x0004_4023 | 17.3.5.1.4/17-22 |
| 0xC54–<br>0xC5C | Reserved | — | — | |
| **Outbound Window 3** | | | | |
| 0xC60 | PEXOTAR3—PCI Express outbound translation address register 3 | R/W | 0x0000_0000 | 17.3.5.1.1/17-20 |
| 0xC64 | PEXOTEAR3—PCI Express outbound translation extended address register 3 | R/W | 0x0000_0000 | 17.3.5.1.2/17-21 |
| 0xC68 | PEXOWBAR3—PCI Express outbound window base address register 3 | R/W | 0x0000_0000 | 17.3.5.1.3/17-22 |
| 0xC6C | Reserved | — | — | |
| 0xC70 | PEXOWAR3—PCI Express outbound window attributes register 3 | R/W | 0x0000_0000 | 17.3.5.1.4/17-22 |
| 0xC74–<br>0xC7C | Reserved | — | — | |
| **Outbound Window 4** | | | | |
| 0xC80 | PEXOTAR4—PCI Express outbound translation address register 4 | R/W | 0x0000_0000 | 17.3.5.1.1/17-20 |
| 0xC84 | PEXOTEAR4—PCI Express outbound translation extended address register 4 | R/W | 0x0000_0000 | 17.3.5.1.2/17-21 |
| 0xC88 | PEXOWBAR4—PCI Express outbound window base address register 4 | R/W | 0x0000_0000 | 17.3.5.1.3/17-22 |
| 0xC8C | Reserved | — | — | |
| 0xC90 | PEXOWAR4—PCI Express outbound window attributes register 4 | R/W | 0x0004_4023 | 17.3.5.1.4/17-22 |
| 0xC94–<br>0xC9C | Reserved | — | — | |
| 0xD14–<br>0xD9C | Reserved | — | — | |
| **Inbound Window 3** | | | | |
| 0xDA0 | PEXITAR3—PCI Express inbound translation address register 3 | R/W | 0x0000_0000 | 17.3.5.2.3/17-26 |
| 0xDA4 | Reserved | — | — | |

Table A-10. PCI Express Controller 1 & 2 Registers (continued)

| PCI Express Controller 1—Block Base Address 0x0_A000<br>PCI Express Controller 2—Block Base Address 0x0_9000<br>PCI Express Controller 3—Block Base Address 0x0_B000 | | | | |
|---|---|---|---|---|
| Offset | Register | Access | Reset | Section/Page |
| 0xDA8 | PEXIWBAR3—PCI Express inbound window base address register 3 | R/W | 0x0000_0000 | 17.3.5.2.4/17-27 |
| 0xDAC | PEXIWBEAR3—PCI Express inbound window base extended address register 3 | R/W | 0x0000_0000 | 17.3.5.2.5/17-27 |
| 0xDB0 | PEXIWAR3—PCI Express inbound window attributes register 3 | R/W | 0x20F4_4023 | 17.3.5.2.6/17-28 |
| 0xDB4–0xDBC | Reserved | — | — | |
| Inbound Window 2 | | | | |
| 0xDC0 | PEXITAR2—PCI Express inbound translation address register 2 | R/W | 0x0000_0000 | 17.3.5.2.3/17-26 |
| 0xDC4 | Reserved | — | — | |
| 0xDC8 | PEXIWBAR2—PCI Express inbound window base address register 2 | R/W | 0x0000_0000 | 17.3.5.2.4/17-27 |
| 0xDCC | PEXIWBEAR2—PCI Express inbound window base extended address register 2 | R/W | 0x0000_0000 | 17.3.5.2.5/17-27 |
| 0xDD0 | PEXIWAR2—PCI Express inbound window attributes register 2 | R/W | 0x20F4_4023 | 17.3.5.2.6/17-28 |
| 0xDD4–0xDDC | Reserved | — | — | |
| Inbound Window 1 | | | | |
| 0xDE0 | PEXITAR1—PCI Express inbound translation address register 1 | R/W | 0x0000_0000 | 17.3.5.2.3/17-26 |
| 0xDE4 | Reserved | — | — | |
| 0xDE8 | PEXIWBAR1—PCI Express inbound window base address register 1 | R/W | 0x0000_0000 | 17.3.5.2.4/17-27 |
| 0xDEC | Reserved | — | — | |
| 0xDF0 | PEXIWAR1—PCI Express inbound window attributes register 1 | R/W | 0x20F4_4023 | 17.3.5.2.6/17-28 |
| 0xDF4–0xDFC | Reserved | — | — | |
| PCI Express Error Management Registers | | | | |
| 0xE00 | PEX_ERR_DR—PCI Express error detect register | w1c | 0x0000_0000 | 17.3.6.1/17-30 |
| 0xE04 | Reserved | — | — | — |
| 0xE08 | PEX_ERR_EN—PCI Express error interrupt enable register | R/W | 0x0000_0000 | 17.3.6.2/17-32 |
| 0xE0C | Reserved | — | — | — |
| 0xE10 | PEX_ERR_DISR—PCI Express error disable register | R/W | 0x0000_0000 | 17.3.6.3/17-34 |
| 0xE14–0xE1C | Reserved | — | — | — |

Table A-10. PCI Express Controller 1 & 2 Registers (continued)

| PCI Express Controller 1—Block Base Address 0x0_A000<br>PCI Express Controller 2—Block Base Address 0x0_9000<br>PCI Express Controller 3—Block Base Address 0x0_B000 | | | | |
|---|---|---|---|---|
| **Offset** | **Register** | **Access** | **Reset** | **Section/Page** |
| 0xE20 | PEX_ERR_CAP_STAT—PCI Express error capture status register | Mixed | 0x0000_0000 | 17.3.6.4/17-36 |
| 0xE24 | Reserved | — | — | — |
| 0xE28 | PEX_ERR_CAP_R0—PCI Express error capture register 0 | R/W | 0x0000_0000 | 17.3.6.5/17-36 |
| 0xE2C | PEX_ERR_CAP_R1—PCI Express error capture register 1 | R/W | 0x0000_0000 | 17.3.6.6/17-38 |
| 0xE30 | PEX_ERR_CAP_R2—PCI Express error capture register 2 | R/W | 0x0000_0000 | 17.3.6.7/17-40 |
| 0xE34 | PEX_ERR_CAP_R3—PCI Express error capture register 3 | R/W | 0x0000_0000 | 17.3.6.8/17-42 |
| 0xE38–0xFFC | Reserved | — | — | |
| **PCI Express Controller 2 Memory-Mapped Registers** | | | | |
| 0x000–0xFFC | PCI Express Controller 2 registers<br>**Note:** All registers defined for PCI Express Controller 1 are also defined for PCI Express Controller 2; the offsets of PCI Express Controller 2 registers are the same except they have a different block base address. | | | |
| **PCI Express Controller 3 Memory-Mapped Registers** | | | | |
| 0x000–0xFFC | PCI Express Controller 3 registers<br>**Note:** All registers defined for PCI Express Controller 1 are also defined for PCI Express Controller 3; the offsets of PCI Express Controller 3 registers are the same except they have a different block base address. | | | |

## A.1.11 General-Purpose I/O (GPIO)

Table A-11. GPIO Registers

| Offset | Register | Access | Reset | Section/Page |
|---|---|---|---|---|
| 0xC00 | GPIO direction register (GPDIR) | R/W | 0x0000_0000 | 22.3.1/22-2 |
| 0xC04 | GPIO open drain register (GPODR) | R/W | 0x0000_0000 | 22.3.2/22-3 |
| 0xC08 | GPIO data register (GPDAT) | R/W | 0x0000_0000 | 22.3.3/22-3 |
| 0xC0C | GPIO interrupt event register (GPIER) | w1c | Undefined | 22.3.4/22-4 |
| 0xC10 | GPIO interrupt mask register (GPIMR) | R/W | 0x0000_0000 | 22.3.5/22-4 |
| 0xC14 | GPIO external interrupt control register (GPICR) | R/W | 0x0000_0000 | 22.3.6/22-5 |

## A.1.12 Serial ATA Controllers

Table A-12. SATA Registers

| SATA Controller 1—Block Base Address 0x1_8000<br>SATA Controller 2—Block Base Address 0x1_9000 | | | | |
|---|---|---|---|---|
| **Offset** | **Register** | **Access** | **Reset** | **Section/Page** |
| **SATA Command Registers** | | | | |
| 0x000 | CQR—Command queue register | R/W | 0x0000_0000 | 19.3.2.1/19-5 |
| 0x008 | CAR—Command active register | R | 0x0000_0000 | 19.3.2.2/19-6 |
| 0x010 | CCR—Command completed register | w1c | 0x0000_0000 | 19.3.2.3/19-6 |
| 0x018 | CER—Command error register | w1c | 0x0000_0000 | 19.3.2.4/19-7 |
| 0x020 | DER—Device error register | w1c | 0x0000_0000 | 19.3.2.5/19-8 |
| 0x024 | CHBA—Command header base address | R/W | 0x0000_0000 | 19.3.2.6/19-8 |
| 0x028 | HStatus—Host status register | w1c | 0x2000_0000 | 19.3.2.7/19-9 |
| 0x02C | HControl—Host control register | Mixed | 0x0000_0100 | 19.3.2.8/19-12 |
| 0x030 | CQPMP—Port number queue register | R/W | 0x0000_0000 | 19.3.2.9/19-13 |
| 0x034 | SIG—Signature register | R | 0xFFFF_FFFF | 19.3.2.10/19-14 |
| 0x038 | ICC—Interrupt coalescing control register | R/W | 0x0100_0000 | 19.3.2.11/19-14 |
| **SATA1 Superset Registers** | | | | |
| 0x100 | SStatus—SATA interface status register | R | 0x0000_0000 | 19.3.3.1/19-15 |
| 0x104 | SError—SATA interface error register | w1c | 0x0000_0000 | 19.3.3.2/19-16 |
| 0x108 | SControl—SATA interface control register | R/W | 0x0000_0300 | 19.3.3.3/19-18 |
| 0x10C | SNotification—SATA interface notification register | w1c | 0x0000_0000 | 19.3.3.4/19-19 |
| **SATA1 Control Status Registers** | | | | |
| 0x140 | TransCfg—Transport layer configuration | R/W | 0x0800_0016 | 19.3.4.1/19-20 |
| 0x144 | TransStatus—Transport layer status | R | 0x0000_0000 | 19.3.4.2/19-21 |
| 0x148 | LinkCfg—Link layer configuration | R/W | 0x0000_FF34 | 19.3.4.3/19-21 |
| 0x14C | LinkCfg1—Link layer configuration1 | R/W | 0x0000_0000 | 19.3.4.4/19-22 |
| 0x150 | LinkCfg2—Link layer configuration2 | R/W | 0x0000_0000 | 19.3.4.5/19-23 |
| 0x154 | LinkStatus—Link layer status | R | 0x0000_0000 | 19.3.4.6/19-23 |
| 0x158 | LinkStatus1—Link layer status1 | R | 0x0000_0000 | 19.3.4.7/19-24 |
| 0x15C | PhyCtrlCfg1—PHY control configuration1 | R/W | 0x0000_3800 | 19.3.4.8/19-26 |
| 0x160 | CommandStatus—Link layer command status | R | 0x0000_0000 | 19.3.4.9/19-27 |
| 0x164–<br>0x17C | Reserved | — | — | — |
| **SATA1 System Control Registers** | | | | |
| 0x410 | SYSPR—System priority register | R/W | 0x0000_0000 | 19.3.5.1/19-28 |

Table A-12. SATA Registers (continued)

| SATA Controller 1—Block Base Address 0x1_8000<br>SATA Controller 2—Block Base Address 0x1_9000 | | | | |
|---|---|---|---|---|
| Offset | Register | Access | Reset | Section/Page |
| 0x40C–0xFFF | Reserved | — | — | — |
| SATA2—Block Base Address: 0x1_9000 | | | | |
| SATA2 has the same memory-mapped registers that are described for SATA1 from 0x1_8000 to 0x1_8FFF except the offsets are from 0x1_9000 to 0x1_9FFF. | | | | |

## A.1.13 L2 Cache/SRAM

Table A-13. L2/SRAM Memory-Mapped Registers

| Offset | Register | Access | Reset | Section/Page |
|---|---|---|---|---|
| L2/SRAM Memory-Mapped Configuration Registers—Block Base Address: 0x2_0000 | | | | |
| 0x000 | L2CTL—L2 control register | R/W | 0x2000_0000 | 6.3.1.1/6-10 |
| 0x010 | L2CEWAR0—L2 cache external write address register 0 | R/W | 0x0000_0000 | 6.3.1.2.1/6-13 |
| 0x014 | L2CEWAREA0—L2 cache external write address register extended address 0 | R/W | 0x0000_0000 | 6.3.1.2.2/6-14 |
| 0x018 | L2CEWCR0—L2 cache external write control register 0 | R/W | 0x0000_0000 | 6.3.1.2.3/6-14 |
| 0x020 | L2CEWAR1—L2 cache external write address register 1 | R/W | 0x0000_0000 | 6.3.1.2.1/6-13 |
| 0x024 | L2CEWAREA1—L2 cache external write address register extended address 1 | R/W | 0x0000_0000 | 6.3.1.2.2/6-14 |
| 0x028 | L2CEWCR1—L2 cache external write control register 1 | R/W | 0x0000_0000 | 6.3.1.2.3/6-14 |
| 0x030 | L2CEWAR2—L2 cache external write address register 2 | R/W | 0x0000_0000 | 6.3.1.2.1/6-13 |
| 0x034 | L2CEWAREA2—L2 cache external write address register extended address 2 | R/W | 0x0000_0000 | 6.3.1.2.2/6-14 |
| 0x038 | L2CEWCR2—L2 cache external write control register 2 | R/W | 0x0000_0000 | 6.3.1.2.3/6-14 |
| 0x040 | L2CEWAR3—L2 cache external write address register 3 | R/W | 0x0000_0000 | 6.3.1.2.1/6-13 |
| 0x044 | L2CEWAREA3—L2 cache external write address register extended address 3 | R/W | 0x0000_0000 | 6.3.1.2.2/6-14 |
| 0x048 | L2CEWCR3—L2 cache external write control register 3 | R/W | 0x0000_0000 | 6.3.1.2.3/6-14 |
| 0x100 | L2SRBAR0—L2 memory-mapped SRAM base address register 0 | R/W | 0x0000_0000 | 6.3.1.3.1/6-16 |
| 0x104 | L2SRBAREA0—L2 memory-mapped SRAM base address register extended address 0 | R/W | 0x0000_0000 | 6.3.1.3.2/6-17 |
| 0x108 | L2SRBAR1—L2 memory-mapped SRAM base address register 1 | R/W | 0x0000_0000 | 6.3.1.3.1/6-16 |
| 0x10C | L2SRBAREA1—L2 memory-mapped SRAM base address register extended address 1 | R/W | 0x0000_0000 | 6.3.1.3.2/6-17 |
| 0xE00 | L2ERRINJHI—L2 error injection mask high register | R/W | 0x0000_0000 | 6.3.1.4.1/6-18 |
| 0xE04 | L2ERRINJLO—L2 error injection mask low register | R/W | 0x0000_0000 | 6.3.1.4.1/6-18 |

**Table A-13. L2/SRAM Memory-Mapped Registers (continued)**

| Offset | Register | Access | Reset | Section/Page |
|---|---|---|---|---|
| 0xE08 | L2ERRINJCTL—L2 error injection tag/ECC control register | R/W | 0x0000_0000 | 6.3.1.4.1/6-18 |
| 0xE20 | L2CAPTDATAHI—L2 error data high capture register | R | 0x0000_0000 | 6.3.1.4.2/6-20 |
| 0xE24 | L2CAPTDATALO—L2 error data low capture register | R | 0x0000_0000 | 6.3.1.4.2/6-20 |
| 0xE28 | L2CAPTECC—L2 error syndrome register | R | 0x0000_0000 | 6.3.1.4.2/6-20 |
| 0xE40 | L2ERRDET—L2 error detect register | w1c | 0x0000_0000 | 6.3.1.4.2/6-20 |
| 0xE44 | L2ERRDIS—L2 error disable register | R/W | 0x0000_0000 | 6.3.1.4.2/6-20 |
| 0xE48 | L2ERRINTEN—L2 error interrupt enable register | R/W | 0x0000_0000 | 6.3.1.4.2/6-20 |
| 0xE4C | L2ERRATTR—L2 error attributes capture register | R/W | 0x0000_0000 | 6.3.1.4.2/6-20 |
| 0xE50 | L2ERRADDRL—L2 error address capture register low | R | 0x0000_0000 | 6.3.1.4.2/6-20 |
| 0xE54 | L2ERRADDRH—L2 error address capture register high | R | 0x0000_0000 | 6.3.1.4.2/6-20 |
| 0xE58 | L2ERRCTL—L2 error control register | R/W | 0x0000_0000 | 6.3.1.4.2/6-20 |

## A.1.14 DMA Controller

## A.1.15 USB Registers

**Table 0-1. USB Interface Memory Map**

| USB Controller 1—Block Base Address 0x2_2000<br>USB Controller 2—Block Base Address 0x2_3000<br>USB Controller 3—Block Base Address 0x2_B000 | | | | |
|---|---|---|---|---|
| **Offset** | **Register** | **Access** | **Reset** | **Section/Page** |
| **USB Controller 1 Registers** | | | | |
| 0x000–0x0FF | Reserved, should be cleared | — | — | — |
| 0x100 | CAPLENGTH—Capability register length | R | 0x40 | 21.3.1.1/21-7 |
| 0x102 | HCIVERSION—Host interface version number | R | 0x0100 | 21.3.1.2/21-7 |
| 0x104 | HCSPARAMS—Host crtl. structural parameters | R | 0x0111_0011 | 21.3.1.3/21-7 |
| 0x108 | HCCPARAMS—Host crtl. capability parameters | R | 0x0000_0006 | 21.3.1.4/21-8 |
| 0x120 | DCIVERSION—Device interface version number | R | 0x0001 | 21.3.1.5/21-9 |
| 0x124 | DCCPARAMS—Device controller parameters | R | 0x0000_0186 | 21.3.1.6/21-10 |
| 0x140 | USBCMD—USB command | Mixed | 0x0008_*n*B*n*0 | 21.3.2.1/21-11 |
| 0x144 | USBSTS—USB status | Mixed | 0x0000_00*n*0 | 21.3.2.2/21-13 |
| 0x148 | USBINTR—USB interrupt enable | R/W | 0x0000_0000 | 21.3.2.3/21-15 |
| 0x14C | FRINDEX—USB frame index | R/W | 0x0000_*nnnn* | 21.3.2.4/21-17 |
| 0x154 | PERIODICLISTBASE—Frame list base address | R/W | 0x*nnnn*_0000 | 21.3.2.6/21-18 |
| | DEVICEADDR—USB device address | R/W | 0x0000_0000 | 21.3.2.7/21-19 |

**Table 0-1. USB Interface Memory Map (continued)**

| **USB Controller 1—Block Base Address 0x2_2000**<br>**USB Controller 2—Block Base Address 0x2_3000**<br>**USB Controller 3—Block Base Address 0x2_B000** | | | | |
|---|---|---|---|---|
| **Offset** | **Register** | **Access** | **Reset** | **Section/Page** |
| 0x158 | ASYNCLISTADDR—Next asynchronous list addr (host mode)[2] | R/W | 0x0000_0000 | 21.3.2.8/21-19 |
| | ENDPOINT ADDR—Address at endpoint list (device mode) | R/W | 0x0000_0000 | 21.3.2.9/21-20 |
| 0x160 | BURSTSIZE—Programmable burst size | R/W | 0x0000_1010 | 21.3.2.10/21-21 |
| 0x164 | TXFILLTUNING—Host TT transmit pre-buffer packet tuning | R/W | 0x0002_0000 | 21.3.2.11/21-21 |
| 0x170 | ULPI VIEWPORT—ULPI Register Access | Mixed | 0x0*n*00_0000 | 21.3.2.12/21-23 |
| 0x180 | CONFIGFLAG—Configured flag register | R | 0x0000_0001 | 21.3.2.13/21-24 |
| 0x184 | PORTSC—Port status/control | Mixed | 0x9C00_000*n* | 21.3.2.14/21-25 |
| 0x1A8 | USBMODE—USB device mode | R/W | 0x0000_0000 | 21.3.2.15/21-29 |
| 0x1AC | ENDPTSETUPSTAT—Endpoint setup status | R/W | 0x0000_0000 | 21.3.2.16/21-30 |
| 0x1B0 | ENDPOINTPRIME—Endpoint initialization | R/W | 0x0000_0000 | 21.3.2.17/21-31 |
| 0x1B4 | ENDPTFLUSH—Endpoint de-initialize | R/W | 0x0000_0000 | 21.3.2.18/21-32 |
| 0x1B8 | ENDPTSTATUS—Endpoint status | R | 0x0000_0000 | 21.3.2.19/21-32 |
| 0x1BC | ENDPTCOMPLETE—Endpoint complete | w1c | 0x0000_0000 | 21.3.2.20/21-33 |
| 0x1C0 | ENDPTCTRL0—Endpoint control 0 | Mixed | 0x0080_0080 | 21.3.2.21/21-33 |
| 0x1C4 | ENDPTCTRL1—Endpoint control 1 | R/W | 0x0000_0000 | 21.3.2.22/21-35 |
| 0x1C8 | ENDPTCTRL2—Endpoint control 2 | R/W | 0x0000_0000 | 21.3.2.22/21-35 |
| 0x1CA | ENDPTCTRL3—Endpoint control 3 | R/W | 0x0000_0000 | 21.3.2.22/21-35 |
| 0x1D0 | ENDPTCTRL4—Endpoint control 4 | R/W | 0x0000_0000 | 21.3.2.22/21-35 |
| 0x1D4 | ENDPTCTRL5—Endpoint control 5 | R/W | 0x0000_0000 | 21.3.2.22/21-35 |
| 0x400 | SNOOP1—Snoop 1 | R/W | 0x0000_0000 | 21.3.2.23/21-36 |
| 0x404 | SNOOP2—Snoop 2 | R/W | 0x0000_0000 | 21.3.2.23/21-36 |
| 0x408 | AGE_CNT_THRESH—Age count threshold | R/W | 0x0000_0000 | 21.3.2.24/21-37 |
| 0x40C | PRI_CTRL—Priority control | R/W | 0x0000_0000 | 21.3.2.25/21-38 |
| 0x410 | SI_CTRL—System interface control | R/W | 0x0000_0000 | 21.3.2.26/21-39 |
| 0x500 | CONTROL—Control | Mixed | 0x0000_0000 | 21.3.2.27/21-39 |
| 0x504–0xFFF | Reserved, should be cleared | — | — | — |
| **USB Controller 2 Registers** | | | | |
| 0x000–0xFFC | USB controller 2 registers<br>**Note:** All registers defined for USB controller 1 are also defined for USB controller 2; the offsets of USB controller 2 registers are the same except they have a different block base address. | | | |

**Table 0-1. USB Interface Memory Map (continued)**

| USB Controller 1—Block Base Address 0x2_2000<br>USB Controller 2—Block Base Address 0x2_3000<br>USB Controller 3—Block Base Address 0x2_B000 | | | | |
|---|---|---|---|---|
| **Offset** | **Register** | **Access** | **Reset** | **Section/Page** |
| **USB Controller 3 Registers** | | | | |
| 0x000–<br>0xFFC | USB controller 3 registers<br>**Note:** All registers defined for USB controller 1 are also defined for USB controller 3; the offsets of USB controller 3 registers are the same except they have a different block base address. | | | |

## A.1.16 eTSEC Registers

**Table A-2. Module Memory Map**

| eTSEC1 Offset | Name[1] | Access [2] | Reset | Section/Page |
|---|---|---|---|---|
| **eTSEC Block Base Address: 0x2_4000 (eTSEC1), 0x2_6000 (eTSEC3)** | | | | |
| **eTSEC General Control and Status Registers** | | | | |
| 0x2_4000 | TSEC_ID*—Controller ID register | R | 0x0124_0000 | 14.5.3.1.1/14-26 |
| 0x2_4004 | TSEC_ID2*—Controller ID register | R | 0x0030_00F0 | 14.5.3.1.2/14-27 |
| 0x2_4008–<br>0x2_400C | Reserved | — | — | — |
| 0x2_4010 | IEVENT—Interrupt event register | w1c | 0x0000_0000 | 14.5.3.1.3/14-27 |
| 0x2_4014 | IMASK—Interrupt mask register | R/W | 0x0000_0000 | 14.5.3.1.4/14-31 |
| 0x2_4018 | EDIS—Error disabled register | R/W | 0x0000_0000 | 14.5.3.1.5/14-33 |
| 0x2_401C | Reserved | — | — | — |
| 0x2_4020 | ECNTRL—Ethernet control register | R/W | 0x0000_0000 | 14.5.3.1.6/14-35 |
| 0x2_4024 | Reserved | — | — | — |
| 0x2_4028 | PTV—Pause time value register | R/W | 0x0000_0000 | 14.5.3.1.7/14-37 |
| 0x2_402C | DMACTRL—DMA control register | R/W | 0x0000_0000 | 14.5.3.1.8/14-38 |
| 0x2_4030 | TBIPA—TBI PHY address register | R/W | 0x0000_0000 | 14.5.3.1.9/14-40 |
| 0x2_4034–<br>0x2_40FC | Reserved | — | — | — |
| **eTSEC Transmit Control and Status Registers** | | | | |
| 0x2_4100 | TCTRL—Transmit control register | R/W | 0x0000_0000 | 14.5.3.2.1/14-40 |
| 0x2_4104 | TSTAT—Transmit status register | w1c | 0x0000_0000 | 14.5.3.2.2/14-42 |
| 0x2_4108 | DFVLAN*—Default VLAN control word | R/W | 0x8100_0000 | 14.5.3.2.3/14-46 |
| 0x2_410C | Reserved | — | — | — |
| 0x2_4110 | TXIC—Transmit interrupt coalescing register | R/W | 0x0000_0000 | 14.5.3.2.4/14-47 |
| 0x2_4114 | TQUEUE*—Transmit queue control register | R/W | 0x0000_8000 | 14.5.3.2.5/14-48 |

### Table A-2. Module Memory Map (continued)

| eTSEC1 Offset | Name[1] | Access [2] | Reset | Section/Page |
|---|---|---|---|---|
| 0x2_4118–<br>0x2_413C | Reserved | — | — | — |
| 0x2_4140 | TR03WT*—TxBD Rings 0–3 round-robin weightings | R/W | 0x0000_0000 | 14.5.3.2.6/14-49 |
| 0x2_4144 | TR47WT*—TxBD Rings 4–7 round-robin weightings | R/W | 0x0000_0000 | 14.5.3.2.7/14-49 |
| 0x2_4148–<br>0x2_417C | Reserved | — | — | — |
| 0x2_4180 | TBDBPH*—Tx data buffer pointer high bits | R/W | 0x0000_0000 | 14.5.3.2.8/14-50 |
| 0x2_4184 | TBPTR0—TxBD pointer for ring 0 | R/W | 0x0000_0000 | 14.5.3.2.9/14-50 |
| 0x2_4188 | Reserved | — | — | — |
| 0x2_418C | TBPTR1*—TxBD pointer for ring 1 | R/W | 0x0000_0000 | 14.5.3.2.9/14-50 |
| 0x2_4190 | Reserved | — | — | — |
| 0x2_4194 | TBPTR2*—TxBD pointer for ring 2 | R/W | 0x0000_0000 | 14.5.3.2.9/14-50 |
| 0x2_4198 | Reserved | — | — | — |
| 0x2_419C | TBPTR3*—TxBD pointer for ring 3 | R/W | 0x0000_0000 | 14.5.3.2.9/14-50 |
| 0x2_41A0 | Reserved | — | — | — |
| 0x2_41A4 | TBPTR4*—TxBD pointer for ring 4 | R/W | 0x0000_0000 | 14.5.3.2.9/14-50 |
| 0x2_41A8 | Reserved | — | — | — |
| 0x2_41AC | TBPTR5*—TxBD pointer for ring 5 | R/W | 0x0000_0000 | 14.5.3.2.9/14-50 |
| 0x2_41B0 | Reserved | — | — | — |
| 0x2_41B4 | TBPTR6*—TxBD pointer for ring 6 | R/W | 0x0000_0000 | 14.5.3.2.9/14-50 |
| 0x2_41B8 | Reserved | — | — | — |
| 0x2_41BC | TBPTR7*—TxBD pointer for ring 7 | R/W | 0x0000_0000 | 14.5.3.2.9/14-50 |
| 0x2_41C0–<br>0x2_41FC | Reserved | — | — | — |
| 0x2_4200 | TBASEH*—TxBD base address high bits | R/W | 0x0000_0000 | 14.5.3.2.10/14-51 |
| 0x2_4204 | TBASE0—TxBD base address of ring 0 | R/W | 0x0000_0000 | 14.5.3.2.11/14-52 |
| 0x2_4208 | Reserved | — | — | — |
| 0x2_420C | TBASE1*—TxBD base address of ring 1 | R/W | 0x0000_0000 | 14.5.3.2.11/14-52 |
| 0x2_4210 | Reserved | — | — | — |
| 0x2_4214 | TBASE2*—TxBD base address of ring 2 | R/W | 0x0000_0000 | 14.5.3.2.11/14-52 |
| 0x2_4218 | Reserved | — | — | — |
| 0x2_421C | TBASE3*—TxBD base address of ring 3 | R/W | 0x0000_0000 | 14.5.3.2.11/14-52 |
| 0x2_4220 | Reserved | — | — | — |
| 0x2_4224 | TBASE4*—TxBD base address of ring 4 | R/W | 0x0000_0000 | 14.5.3.2.11/14-52 |
| 0x2_4228 | Reserved | — | — | — |

**Table A-2. Module Memory Map (continued)**

| eTSEC1 Offset | Name[1] | Access [2] | Reset | Section/Page |
|---|---|---|---|---|
| 0x2_422C | TBASE5*—TxBD base address of ring 5 | R/W | 0x0000_0000 | 14.5.3.2.11/14-52 |
| 0x2_4230 | Reserved | — | — | — |
| 0x2_4234 | TBASE6*—TxBD base address of ring 6 | R/W | 0x0000_0000 | 14.5.3.2.11/14-52 |
| 0x2_4238 | Reserved | — | — | — |
| 0x2_423C | TBASE7*—TxBD base address of ring 7 | R/W | 0x0000_0000 | 14.5.3.2.11/14-52 |
| 0x2_4240–0x2_427C | Reserved | — | — | — |
| 0x2_4280 | TMR_TXTS1_ID* - Tx time stamp identification tag (set 1) | R/W | 0x0000_0000 | 14.5.3.2.12/14-52 |
| 0x2_4284 | TMR_TXTS2_ID* - Tx time stamp identification tag (set 2) | R/W | 0x0000_0000 | 14.5.3.2.12/14-52 |
| 0x2_4288–0x2_42BC | Reserved | — | — | — |
| 0x2_42C0 | TMR_TXTS1_H* - Tx time stamp high (set 1) | R/W | 0x0000_0000 | 14.5.3.2.13/14-53 |
| 0x2_42C4 | TMR_TXTS1_L* - Tx time stamp high (set 1) | R/W | 0x0000_0000 | 14.5.3.2.13/14-53 |
| 0x2_42C8 | TMR_TXTS2_H* - Tx time stamp high (set 2) | R/W | 0x0000_0000 | 14.5.3.2.13/14-53 |
| 0x2_42CC | TMR_TXTS2_L* - Tx time stamp high (set 2) | R/W | 0x0000_0000 | 14.5.3.2.13/14-53 |
| 0x2_42D0–0x2_42FC | Reserved | — | — | — |
| **eTSEC Receive Control and Status Registers** | | | | |
| 0x2_4300 | RCTRL—Receive control register | R/W | 0x0000_0000 | 14.5.3.3.1/14-54 |
| 0x2_4304 | RSTAT—Receive status register | w1c | 0x0000_0000 | 14.5.3.3.2/14-56 |
| 0x2_4308–0x2_430C | Reserved | — | — | — |
| 0x2_4310 | RXIC—Receive interrupt coalescing register | R/W | 0x0000_0000 | 14.5.3.3.3/14-59 |
| 0x2_4314 | RQUEUE*—Receive queue control register. | R/W | 0x0080_0080 | 14.5.3.3.4/14-60 |
| 0x2_4318–0x2_432C | Reserved | — | — | — |
| 0x2_4330 | RBIFX*—Receive bit field extract control register | R/W | 0x0000_0000 | 14.5.3.3.5/14-61 |
| 0x2_4334 | RQFAR*—Receive queue filing table address register | R/W | 0x0000_0000 | 14.5.3.3.6/14-63 |
| 0x2_4338 | RQFCR*—Receive queue filing table control register | R/W | 0x*nnnn_nnnn* | 14.5.3.3.7/14-63 |
| 0x2_433C | RQFPR*—Receive queue filing table property register | R/W | 0x*nnnn_nnnn* | 14.5.3.3.8/14-65 |
| 0x2_4340 | MRBLR—Maximum receive buffer length register | R/W | 0x0000_0000 | 14.5.3.3.9/14-68 |
| 0x2_4344–0x2_437C | Reserved | — | — | — |
| 0x2_4380 | RBDBPH*—Rx data buffer pointer high bits | R/W | 0x0000_0000 | 14.5.3.3.10/14-68 |
| 0x2_4384 | RBPTR0—RxBD pointer for ring 0 | R/W | 0x0000_0000 | 14.5.3.3.11/14-69 |
| 0x2_4388 | Reserved | — | — | — |

Table A-2. Module Memory Map (continued)

| eTSEC1 Offset | Name[1] | Access [2] | Reset | Section/Page |
|---|---|---|---|---|
| 0x2_438C | RBPTR1*—RxBD pointer for ring 1 | R/W | 0x0000_0000 | 14.5.3.3.11/14-69 |
| 0x2_4390 | Reserved | — | — | — |
| 0x2_4394 | RBPTR2*—RxBD pointer for ring 2 | R/W | 0x0000_0000 | 14.5.3.3.11/14-69 |
| 0x2_4398 | Reserved | — | — | — |
| 0x2_439C | RBPTR3*—RxBD pointer for ring 3 | R/W | 0x0000_0000 | 14.5.3.3.11/14-69 |
| 0x2_43A0 | Reserved | — | — | — |
| 0x2_43A4 | RBPTR4*—RxBD pointer for ring 4 | R/W | 0x0000_0000 | 14.5.3.3.11/14-69 |
| 0x2_43A8 | Reserved | — | — | — |
| 0x2_43AC | RBPTR5*—RxBD pointer for ring 5 | R/W | 0x0000_0000 | 14.5.3.3.11/14-69 |
| 0x2_43B0 | Reserved | — | — | — |
| 0x2_43B4 | RBPTR6*—RxBD pointer for ring 6 | R/W | 0x0000_0000 | 14.5.3.3.11/14-69 |
| 0x2_43B8 | Reserved | — | — | — |
| 0x2_43BC | RBPTR7*—RxBD pointer for ring 7 | R/W | 0x0000_0000 | 14.5.3.3.11/14-69 |
| 0x2_43C0–0x2_43FC | Reserved | — | — | — |
| 0x2_4400 | RBASEH*—RxBD base address high bits | R/W | 0x0000_0000 | 14.5.3.3.12/14-70 |
| 0x2_4404 | RBASE0—RxBD base address of ring 0 | R/W | 0x0000_0000 | 14.5.3.3.13/14-70 |
| 0x2_4408 | Reserved | — | — | — |
| 0x2_440C | RBASE1*—RxBD base address of ring 1 | R/W | 0x0000_0000 | 14.5.3.3.13/14-70 |
| 0x2_4410 | Reserved | — | — | — |
| 0x2_4414 | RBASE2*—RxBD base address of ring 2 | R/W | 0x0000_0000 | 14.5.3.3.13/14-70 |
| 0x2_4418 | Reserved | — | — | — |
| 0x2_441C | RBASE3*—RxBD base address of ring 3 | R/W | 0x0000_0000 | 14.5.3.3.13/14-70 |
| 0x2_4420 | Reserved | — | — | — |
| 0x2_4424 | RBASE4*—RxBD base address of ring 4 | R/W | 0x0000_0000 | 14.5.3.3.13/14-70 |
| 0x2_4428 | Reserved | — | — | — |
| 0x2_442C | RBASE5*—RxBD base address of ring 5 | R/W | 0x0000_0000 | 14.5.3.3.13/14-70 |
| 0x2_4430 | Reserved | — | — | — |
| 0x2_4434 | RBASE6*—RxBD base address of ring 6 | R/W | 0x0000_0000 | 14.5.3.3.13/14-70 |
| 0x2_4438 | Reserved | — | — | — |
| 0x2_443C | RBASE7*—RxBD base address of ring 7 | R/W | 0x0000_0000 | 14.5.3.3.13/14-70 |
| 0x2_4440–0x2_44BC | Reserved | — | — | — |
| 0x2_44C0 | TMR_RXTS_H* - Rx timer time stamp register high | R/W | 0x0000_0000 | 14.5.3.3.14/14-71 |
| 0x2_44C4 | TMR_RXTS_L* - Rx timer time stamp register low | R/W | 0x0000_0000 | 14.5.3.3.14/14-71 |

### Table A-2. Module Memory Map (continued)

| eTSEC1 Offset | Name[1] | Access [2] | Reset | Section/Page |
|---|---|---|---|---|
| 0x2_44C8–<br>0x2_44FC | Reserved | — | — | — |
| | **eTSEC MAC Registers** | | | |
| 0x2_4500 | MACCFG1—MAC configuration register 1 | R/W | 0x0000_0000 | 14.5.3.5.1/14-74 |
| 0x2_4504 | MACCFG2—MAC configuration register 2 | R/W | 0x0000_7000 | 14.5.3.5.2/14-76 |
| 0x2_4508 | IPGIFG—Inter-packet/inter-frame gap register | R/W | 0x4060_5060 | 14.5.3.5.3/14-78 |
| 0x2_450C | HAFDUP—Half-duplex control | R/W | 0x00A1_F037 | 14.5.3.5.4/14-79 |
| 0x2_4510 | MAXFRM—Maximum frame length | R/W | 0x0000_0600 | 14.5.3.5.5/14-80 |
| 0x2_4514–<br>0x2_451C | Reserved | — | — | — |
| 0x2_4520 | MIIMCFG—MII management configuration | R/W | 0x0000_0007 | 14.5.3.5.6/14-80 |
| 0x2_4524 | MIIMCOM—MII management command | R/W | 0x0000_0000 | 14.5.3.5.7/14-81 |
| 0x2_4528 | MIIMADD—MII management address | R/W | 0x0000_0000 | 14.5.3.5.8/14-82 |
| 0x2_452C | MIIMCON—MII management control | WO | 0x0000_0000 | 14.5.3.5.9/14-82 |
| 0x2_4530 | MIIMSTAT—MII management status | R | 0x0000_0000 | 14.5.3.5.10/14-83 |
| 0x2_4534 | MIIMIND—MII management indicator | R | 0x0000_0000 | 14.5.3.5.11/14-83 |
| 0x2_4538 | Reserved | — | — | — |
| 0x2_453C | IFSTAT—Interface status | R | 0x0000_0000 | 14.5.3.5.12/14-84 |
| 0x2_4540 | MACSTNADDR1—MAC station address register 1 | R/W | 0x0000_0000 | 14.5.3.5.13/14-84 |
| 0x2_4544 | MACSTNADDR2—MAC station address register 2 | R/W | 0x0000_0000 | 14.5.3.5.14/14-85 |
| 0x2_4548 | MAC01ADDR1*—MAC exact match address 1, part 1 | R/W | 0x0000_0000 | 14.5.3.5.15/14-86<br>14.5.3.5.16/14-86 |
| 0x2_454C | MAC01ADDR2*—MAC exact match address 1, part 2 | R/W | 0x0000_0000 | |
| 0x2_4550 | MAC02ADDR1*—MAC exact match address 2, part 1 | R/W | 0x0000_0000 | |
| 0x2_4554 | MAC02ADDR2*—MAC exact match address 2, part 2 | R/W | 0x0000_0000 | |
| 0x2_4558 | MAC03ADDR1*—MAC exact match address 3, part 1 | R/W | 0x0000_0000 | |
| 0x2_455C | MAC03ADDR2*—MAC exact match address 3, part 2 | R/W | 0x0000_0000 | |
| 0x2_4560 | MAC04ADDR1*—MAC exact match address 4, part 1 | R/W | 0x0000_0000 | |
| 0x2_4564 | MAC04ADDR2*—MAC exact match address 4, part 2 | R/W | 0x0000_0000 | |
| 0x2_4568 | MAC05ADDR1*—MAC exact match address 5, part 1 | R/W | 0x0000_0000 | |
| 0x2_456C | MAC05ADDR2*—MAC exact match address 5, part 2 | R/W | 0x0000_0000 | |

Table A-2. Module Memory Map (continued)

| eTSEC1 Offset | Name[1] | Access [2] | Reset | Section/Page |
|---|---|---|---|---|
| 0x2_4570 | MAC06ADDR1*—MAC exact match address 6, part 1 | R/W | 0x0000_0000 | 14.5.3.5.15/14-86<br>14.5.3.5.16/14-86 |
| 0x2_4574 | MAC06ADDR2*—MAC exact match address 6, part 2 | R/W | 0x0000_0000 | |
| 0x2_4578 | MAC07ADDR1*—MAC exact match address 7, part 1 | R/W | 0x0000_0000 | |
| 0x2_457C | MAC07ADDR2*—MAC exact match address 7, part 2 | R/W | 0x0000_0000 | |
| 0x2_4580 | MAC08ADDR1*—MAC exact match address 8, part 1 | R/W | 0x0000_0000 | |
| 0x2_4584 | MAC08ADDR2*—MAC exact match address 8, part 2 | R/W | 0x0000_0000 | |
| 0x2_4588 | MAC09ADDR1*—MAC exact match address 9, part 1 | R/W | 0x0000_0000 | |
| 0x2_458C | MAC09ADDR2*—MAC exact match address 9, part 2 | R/W | 0x0000_0000 | |
| 0x2_4590 | MAC10ADDR1*—MAC exact match address 10, part 1 | R/W | 0x0000_0000 | |
| 0x2_4594 | MAC10ADDR2*—MAC exact match address 10, part 2 | R/W | 0x0000_0000 | |
| 0x2_4598 | MAC11ADDR1*—MAC exact match address 11, part 1 | R/W | 0x0000_0000 | |
| 0x2_459C | MAC11ADDR2*—MAC exact match address 11, part 2 | R/W | 0x0000_0000 | |
| 0x2_45A0 | MAC12ADDR1*—MAC exact match address 12, part 1 | R/W | 0x0000_0000 | |
| 0x2_45A4 | MAC12ADDR2*—MAC exact match address 12, part 2 | R/W | 0x0000_0000 | |
| 0x2_45A8 | MAC13ADDR1*—MAC exact match address 13, part 1 | R/W | 0x0000_0000 | |
| 0x2_45AC | MAC13ADDR2*—MAC exact match address 13, part 2 | R/W | 0x0000_0000 | |
| 0x2_45B0 | MAC14ADDR1*—MAC exact match address 14, part 1 | R/W | 0x0000_0000 | |
| 0x2_45B4 | MAC14ADDR2*—MAC exact match address 14, part 2 | R/W | 0x0000_0000 | |
| 0x2_45B8 | MAC15ADDR1*—MAC exact match address 15, part 1 | R/W | 0x0000_0000 | |
| 0x2_45BC | MAC15ADDR2*—MAC exact match address 15, part 2 | R/W | 0x0000_0000 | |
| 0x2_45C0–0x2_467C | Reserved | — | — | — |
| **eTSEC Transmit and Receive Counters** | | | | |
| 0x2_4680 | TR64—Transmit and receive 64-byte frame counter | R/W | 0x0000_0000 | 14.5.3.6.1/14-88 |
| 0x2_4684 | TR127—Transmit and receive 65- to 127-byte frame counter | R/W | 0x0000_0000 | 14.5.3.6.2/14-88 |
| 0x2_4688 | TR255—Transmit and receive 128- to 255-byte frame counter | R/W | 0x0000_0000 | 14.5.3.6.3/14-89 |
| 0x2_468C | TR511—Transmit and receive 256- to 511-byte frame counter | R/W | 0x0000_0000 | 14.5.3.6.4/14-89 |
| 0x2_4690 | TR1K—Transmit and receive 512- to 1023-byte frame counter | R/W | 0x0000_0000 | 14.5.3.6.5/14-90 |
| 0x2_4694 | TRMAX—Transmit and receive 1024- to 1518-byte frame counter | R/W | 0x0000_0000 | 14.5.3.6.6/14-90 |
| 0x2_4698 | TRMGV—Transmit and receive 1519- to 1522-byte good VLAN frame count | R/W | 0x0000_0000 | 14.5.3.6.7/14-91 |
| **eTSEC Receive Counters** | | | | |
| 0x2_469C | RBYT—Receive byte counter | R/W | 0x0000_0000 | 14.5.3.6.8/14-91 |
| 0x2_46A0 | RPKT—Receive packet counter | R/W | 0x0000_0000 | 14.5.3.6.9/14-92 |
| 0x2_46A4 | RFCS—Receive FCS error counter | R/W | 0x0000_0000 | 14.5.3.6.10/14-92 |
| 0x2_46A8 | RMCA—Receive multicast packet counter | R/W | 0x0000_0000 | 14.5.3.6.11/14-93 |

### Table A-2. Module Memory Map (continued)

| eTSEC1 Offset | Name[1] | Access [2] | Reset | Section/Page |
|---|---|---|---|---|
| 0x2_46AC | RBCA—Receive broadcast packet counter | R/W | 0x0000_0000 | 14.5.3.6.12/14-93 |
| 0x2_46B0 | RXCF—Receive control frame packet counter | R/W | 0x0000_0000 | 14.5.3.6.13/14-94 |
| 0x2_46B4 | RXPF—Receive PAUSE frame packet counter | R/W | 0x0000_0000 | 14.5.3.6.14/14-94 |
| 0x2_46B8 | RXUO—Receive unknown OP code counter | R/W | 0x0000_0000 | 14.5.3.6.15/14-95 |
| 0x2_46BC | RALN—Receive alignment error counter | R/W | 0x0000_0000 | 14.5.3.6.16/14-95 |
| 0x2_46C0 | RFLR—Receive frame length error counter | R/W | 0x0000_0000 | 14.5.3.6.17/14-96 |
| 0x2_46C4 | RCDE—Receive code error counter | R/W | 0x0000_0000 | 14.5.3.6.18/14-96 |
| 0x2_46C8 | RCSE—Receive carrier sense error counter | R/W | 0x0000_0000 | 14.5.3.6.19/14-97 |
| 0x2_46CC | RUND—Receive undersize packet counter | R/W | 0x0000_0000 | 14.5.3.6.20/14-97 |
| 0x2_46D0 | ROVR—Receive oversize packet counter | R/W | 0x0000_0000 | 14.5.3.6.21/14-98 |
| 0x2_46D4 | RFRG—Receive fragments counter | R/W | 0x0000_0000 | 14.5.3.6.22/14-98 |
| 0x2_46D8 | RJBR—Receive jabber counter | R/W | 0x0000_0000 | 14.5.3.6.23/14-99 |
| 0x2_46DC | RDRP—Receive drop counter | R/W | 0x0000_0000 | 14.5.3.6.24/14-99 |
| **eTSEC Transmit Counters** | | | | |
| 0x2_46E0 | TBYT—Transmit byte counter | R/W | 0x0000_0000 | 14.5.3.6.25/14-100 |
| 0x2_46E4 | TPKT—Transmit packet counter | R/W | 0x0000_0000 | 14.5.3.6.26/14-100 |
| 0x2_46E8 | TMCA—Transmit multicast packet counter | R/W | 0x0000_0000 | 14.5.3.6.27/14-101 |
| 0x2_46EC | TBCA—Transmit broadcast packet counter | R/W | 0x0000_0000 | 14.5.3.6.28/14-101 |
| 0x2_46F0 | TXPF—Transmit PAUSE control frame counter | R/W | 0x0000_0000 | 14.5.3.6.29/14-102 |
| 0x2_46F4 | TDFR—Transmit deferral packet counter | R/W | 0x0000_0000 | 14.5.3.6.30/14-102 |
| 0x2_46F8 | TEDF—Transmit excessive deferral packet counter | R/W | 0x0000_0000 | 14.5.3.6.31/14-103 |
| 0x2_46FC | TSCL—Transmit single collision packet counter | R/W | 0x0000_0000 | 14.5.3.6.32/14-103 |
| 0x2_4700 | TMCL—Transmit multiple collision packet counter | R/W | 0x0000_0000 | 14.5.3.6.33/14-104 |
| 0x2_4704 | TLCL—Transmit late collision packet counter | R/W | 0x0000_0000 | 14.5.3.6.34/14-104 |
| 0x2_4708 | TXCL—Transmit excessive collision packet counter | R/W | 0x0000_0000 | 14.5.3.6.35/14-105 |
| 0x2_470C | TNCL—Transmit total collision counter | R/W | 0x0000_0000 | 14.5.3.6.36/14-105 |
| 0x2_4710 | Reserved | — | — | — |
| 0x2_4714 | TDRP—Transmit drop frame counter | R/W | 0x0000_0000 | 14.5.3.6.37/14-106 |
| 0x2_4718 | TJBR—Transmit jabber frame counter | R/W | 0x0000_0000 | 14.5.3.6.38/14-106 |
| 0x2_471C | TFCS—Transmit FCS error counter | R/W | 0x0000_0000 | 14.5.3.6.39/14-107 |
| 0x2_4720 | TXCF—Transmit control frame counter | R/W | 0x0000_0000 | 14.5.3.6.40/14-107 |
| 0x2_4724 | TOVR—Transmit oversize frame counter | R/W | 0x0000_0000 | 14.5.3.6.41/14-108 |
| 0x2_4728 | TUND—Transmit undersize frame counter | R/W | 0x0000_0000 | 14.5.3.6.42/14-108 |
| 0x2_472C | TFRG—Transmit fragments frame counter | R/W | 0x0000_0000 | 14.5.3.6.43/14-109 |

Table A-2. Module Memory Map (continued)

| eTSEC1 Offset | Name[1] | Access [2] | Reset | Section/Page |
|---|---|---|---|---|
| **eTSEC Counter Control and TOE Statistics Registers** | | | | |
| 0x2_4730 | CAR1—Carry register one register[3] | R | 0x0000_0000 | 14.5.3.6.44/14-109 |
| 0x2_4734 | CAR2—Carry register two register [3] | R | 0x0000_0000 | 14.5.3.6.45/14-111 |
| 0x2_4738 | CAM1—Carry register one mask register | R/W | 0xFE03_FFFF | 14.5.3.6.46/14-112 |
| 0x2_473C | CAM2—Carry register two mask register | R/W | 0x000F_FFFD | 14.5.3.6.47/14-113 |
| 0x2_4740 | RREJ*—Receive filer rejected packet counter | R/W | 0x0000_0000 | 14.5.3.6.48/14-114 |
| 0x2_4744–<br>0x2_47FC | Reserved | — | — | — |
| **Hash Function Registers** | | | | |
| 0x2_4800 | IGADDR0—Individual/group address register 0 | R/W | 0x0000_0000 | 14.5.3.7.1/14-115 |
| 0x2_4804 | IGADDR1—Individual/group address register 1 | R/W | 0x0000_0000 | |
| 0x2_4808 | IGADDR2—Individual/group address register 2 | R/W | 0x0000_0000 | |
| 0x2_480C | IGADDR3—Individual/group address register 3 | R/W | 0x0000_0000 | |
| 0x2_4810 | IGADDR4—Individual/group address register 4 | R/W | 0x0000_0000 | |
| 0x2_4814 | IGADDR5—Individual/group address register 5 | R/W | 0x0000_0000 | |
| 0x2_4818 | IGADDR6—Individual/group address register 6 | R/W | 0x0000_0000 | |
| 0x2_481C | IGADDR7—Individual/group address register 7 | R/W | 0x0000_0000 | |
| 0x2_4820–<br>0x2_487C | Reserved | — | — | — |
| 0x2_4880 | GADDR0—Group address register 0 | R/W | 0x0000_0000 | 14.5.3.7.2/14-116 |
| 0x2_4884 | GADDR1—Group address register 1 | R/W | 0x0000_0000 | |
| 0x2_4888 | GADDR2—Group address register 2 | R/W | 0x0000_0000 | |
| 0x2_488C | GADDR3—Group address register 3 | R/W | 0x0000_0000 | |
| 0x2_4890 | GADDR4—Group address register 4 | R/W | 0x0000_0000 | |
| 0x2_4894 | GADDR5—Group address register 5 | R/W | 0x0000_0000 | |
| 0x2_4898 | GADDR6—Group address register 6 | R/W | 0x0000_0000 | |
| 0x2_489C | GADDR7—Group address register 7 | R/W | 0x0000_0000 | |
| 0x2_48A0–<br>0x2_49FC | Reserved | — | — | — |
| **eTSEC FIFO Control Registers** | | | | |
| 0x2_4A00 | FIFOCFG*—FIFO interface configuration register | R/W | 0x0000_00C0 | 14.5.3.8.1/14-116 |
| 0x2_4A04–<br>0x2_4AFC | Reserved | — | — | — |
| **eTSEC DMA Attribute Registers** | | | | |
| 0x2_4B00–<br>0x2_4BF4 | Reserved | — | — | — |

Table A-2. Module Memory Map (continued)

| eTSEC1 Offset | Name[1] | Access [2] | Reset | Section/Page |
|---|---|---|---|---|
| 0x2_4BF8 | ATTR—Attribute register | R/W | 0x0000_0000 | 14.5.3.9.1/14-118 |
| 0x2_4BFC | ATTRELI*—Attribute extract length and extract index register | R/W | 0x0000_0000 | 14.5.3.9.2/14-119 |
| **eTSEC Lossless Flow Control Registers** | | | | |
| 0x2_4C00 | RQPRM0*—Receive Queue Parameters register 0 | R/W | 0x0000_0000 | 14.5.3.10.1/14-120 |
| 0x2_4C04 | RQPRM1*—Receive Queue Parameters register 1 | R/W | 0x0000_0000 | |
| 0x2_4C08 | RQPRM2*—Receive Queue Parameters register 2 | R/W | 0x0000_0000 | |
| 0x2_4C0C | RQPRM3*—Receive Queue Parameters register 3 | R/W | 0x0000_0000 | |
| 0x2_4C10 | RQPRM4*—Receive Queue Parameters register 4 | R/W | 0x0000_0000 | |
| 0x2_4C14 | RQPRM5*—Receive Queue Parameters register 5 | R/W | 0x0000_0000 | |
| 0x2_4C18 | RQPRM6*—Receive Queue Parameters register 6 | R/W | 0x0000_0000 | |
| 0x2_4C1C | RQPRM7*—Receive Queue Parameters register 7 | R/W | 0x0000_0000 | |
| 0x2_4C20–<br>0x2_4C40 | Reserved | — | — | — |
| 0x2_4C44 | RFBPTR0*—Last Free RxBD pointer for ring 0 | R/W | 0x0000_0000 | 14.5.3.10.2/14-121 |
| 0x2_4C48 | Reserved | — | — | — |
| 0x2_4C4C | RFBPTR1*—Last Free RxBD pointer for ring 1 | R/W | 0x0000_0000 | 14.5.3.10.2/14-121 |
| 0x2_4C50 | Reserved | — | — | — |
| 0x2_4C54 | RFBPTR2*—Last Free RxBD pointer for ring 2 | R/W | 0x0000_0000 | 14.5.3.10.2/14-121 |
| 0x2_4C58 | Reserved | — | — | — |
| 0x2_4C5C | RFBPTR3*—Last Free RxBD pointer for ring 3 | R/W | 0x0000_0000 | 14.5.3.10.2/14-121 |
| 0x2_4C60 | Reserved | — | — | — |
| 0x2_4C64 | RFBPTR4*—Last Free RxBD pointer for ring 4 | R/W | 0x0000_0000 | 14.5.3.10.2/14-121 |
| 0x2_4C68 | Reserved | — | — | — |
| 0x2_4C6C | RFBPTR5*—Last Free RxBD pointer for ring 5 | R/W | 0x0000_0000 | 14.5.3.10.2/14-121 |
| 0x2_4C70 | Reserved | — | — | — |
| 0x2_4C74 | RFBPTR6*—Last Free RxBD pointer for ring 6 | R/W | 0x0000_0000 | 14.5.3.10.2/14-121 |
| 0x2_4C78 | Reserved | — | — | — |
| 0x2_4C7C | RFBPTR7*—Last Free RxBD pointer for ring 7 | R/W | 0x0000_0000 | 14.5.3.10.2/14-121 |
| **eTSEC Future Expansion Space** | | | | |
| 0x2_4CC0<br>–<br>0x2_4D94 | Reserved | — | — | — |
| **eTSEC IEEE 1588 Registers** | | | | |
| 0x2_4E00 | TMR_CTRL* - Timer control register | R/W | 0x0001_0001 | 14.5.3.11.1/14-122 |
| 0x2_4E04 | TMR_TEVENT* - time stamp event register | W1C | 0x0000_0000 | 14.5.3.11.2/14-124 |

Table A-2. Module Memory Map (continued)

| eTSEC1 Offset | Name[1] | Access [2] | Reset | Section/Page |
|---|---|---|---|---|
| 0x2_4E08 | TMR_TEMASK* - Timer event mask register | R/W | 0x0000_0000 | 14.5.3.11.3/14-125 |
| 0x2_4E0C | TMR_PEVENT* - time stamp event register | R/W | 0x0000_0000 | 14.5.3.11.4/14-126 |
| 0x2_4E10 | TMR_PEMASK* - Timer event mask register | R/W | 0x0000_0000 | 14.5.3.11.5/14-127 |
| 0x2_4E14 | TMR_STAT* - time stamp status register | R/W | 0x0000_0000 | 14.5.3.11.6/14-128 |
| 0x2_4E18 | TMR_CNT_H* - timer counter high register | R/W | 0x0000_0000 | 14.5.3.11.7/14-128 |
| 0x2_4E1C | TMR_CNT_L* - timer counter low register | R/W | 0x0000_0000 | 14.5.3.11.7/14-128 |
| 0x2_4E20 | TMR_ADD* - Timer drift compensation addend register | R/W | 0x0000_0000 | 14.5.3.11.8/14-129 |
| 0x2_4E24 | TMR_ACC* - Timer accumulator register | R/W | 0x0000_0000 | 14.5.3.11.9/14-130 |
| 0x2_4E28 | TMR_PRSC* -Timer prescale | R/W | 0x0000_0002 | 14.5.3.11.10/14-130 |
| 0x2_4E2C | Reserved | — | — | — |
| 0x2_4E30 | TMROFF_H* - Timer offset high | R/W | 0x0000_0000 | 14.5.3.11.11/14-131 |
| 0x2_4E34 | TMROFF_L* - Timer offset low | R/W | 0x0000_0000 | 14.5.3.11.11/14-131 |
| 0x2_4E40 | TMR_ALARM1_H* - Timer alarm 1 high register | R/W | 0xFFFF_FFFF | 14.5.3.11.12/14-131 |
| 0x2_4E44 | TMR_ALARM1_L* - Timer alarm 1 high register | R/W | 0xFFFF_FFFF | |
| 0x2_4E48 | TMR_ALARM2_H* - Timer alarm 2 high register | R/W | 0xFFFF_FFFF | |
| 0x2_4E4C | TMR_ALARM2_L* - Timer alarm 2 high register | R/W | 0xFFFF_FFFF | |
| 0x2_4E50–<br>0x2_4E7C | Reserved | — | — | — |
| 0x2_4E80 | TMR_FIPER1* - Timer fixed period interval | R/W | 0xFFFF_FFFF | 14.5.3.11.13/14-132 |
| 0x2_4E84 | TMR_FIPER2* - Timer fixed period interval | R/W | 0xFFFF_FFFF | |
| 0x2_4E88 | TMR_FIPER*3 - Timer fixed period interval | R/W | 0xFFFF_FFFF | |
| 0x2_4EA0 | TMR_ETTS1_H* - Time stamp of general purpose external trigger | R/W | 0x0000_0000 | 14.5.3.11.14/14-133 |
| 0x2_4EA4 | TMR_ETTS1_L* - Time stamp of general purpose external trigger | R/W | 0x0000_0000 | |
| 0x2_4EA8 | TMR_ETTS2_H* - Time stamp of general purpose external trigger | R/W | 0x0000_0000 | |
| 0x2_4EAC | TMR_ETTS2_L* - Time stamp of general purpose external trigger | R/W | 0x0000_0000 | |
| 0x2_4EB0<br>–<br>0x2_4FFF | Reserved | — | — | |
| Other eTSECs | | | | |
| 0x2_6000–<br>0x2_6FFF | eTSEC3 REGISTERS[4] | | | |

[1] Registers denoted * are new to the enhanced TSEC and not supported by PowerQUICC III TSECs.

[2] Key: R = read only, WO = write only, R/W = read and write, LH = latches high, SC = self-clearing.

[3] Cleared on read.

[4] eTSEC3 has the same memory-mapped registers that are described for eTSEC1 from 0x 2_4000 to 0x2_4FFF, except the offsets are from 0x 2_6000 to 0x2_6FFF.

## A.1.17 eSDHC Registers

**Table 1-3. eSDHC Memory Map**

| eSDHC Registers—Block Base Address 0x2_E000 | | | | |
|---|---|---|---|---|
| **Offset** | **Register** | **Access** | **Reset** | **Section/Page** |
| 0x000 | DMA system address (DSADDR) | R/W | 0x0000_0008 | 20.4.1/20-6 |
| 0x004 | Block attributes (BLKATTR) | R/W | 0x0000_0008 | 20.4.2/20-6 |
| 0x008 | Command argument (CMDARG) | R/W | 0x0000_0000 | 20.4.3/20-7 |
| 0x00C | Command transfer type (XFERTYP) | R/W | 0x0000_0000 | 20.4.4/20-8 |
| 0x010 | Command response0 (CMDRSP0) | R | 0x0000_0000 | 20.4.5/20-11 |
| 0x014 | Command response1 (CMDRSP1) | R | 0x0000_0000 | 20.4.5/20-11 |
| 0x018 | Command response2 (CMDRSP2) | R | 0x0000_0000 | 20.4.5/20-11 |
| 0x01C | Command response3 (CMDRSP3) | R | 0x0000_0000 | 20.4.5/20-11 |
| 0x020 | Data buffer access port (DATPORT) | R/W | 0x0000_0000 | 20.4.6/20-12 |
| 0x024 | Present state (PRSSTAT) | R | 0xFF80_0000 | 20.4.7/20-13 |
| 0x028 | Protocol control (PROCTL) | R/W | 0x0000_0020 | 20.4.8/20-17 |
| 0x02C | System control (SYSCTL) | Mixed | 0x0000_8000 | 20.4.9/20-19 |
| 0x030 | Interrupt status (IRQSTAT) | w1c | 0x0000_0000 | 20.4.10/20-22 |
| 0x034 | Interrupt status enable (IRQSTATEN) | R/W | 0x117F_013F | 20.4.11/20-26 |
| 0x038 | Interrupt signal enable (IRQSIGEN) | R/W | 0x0000_0000 | 20.4.12/20-29 |
| 0x03C | Auto CMD12 status (AUTOC12ERR) | R | 0x0000_0000 | 20.4.13/20-31 |
| 0x040 | Host controller capabilities (HOSTCAPBLT) | R | 0x01E3_0000 | 20.4.14/20-33 |
| 0x044[1] | Watermark level (WML) | R/W | 0x0010_0010 | 20.4.15/20-34 |
| 0x050 | Force event (FEVT) | W | 0x0000_0000 | 20.4.16/20-34 |
| 0x0FC | Host controller version (HOSTVER) | R | 0x0000_0001 | 20.4.17/20-36 |
| 0x40C | DMA control register (DCR) | R/W | 0x0000_0000 | 20.4.18/20-37 |

[1] The addresses following 0x044, except 0x050, 0x0FC and 0x40C, are reserved and read as all 0s. Writes to these registers are ignored.

## A.1.18 SEC Registers

Table 1-4. SEC Address Map

| Byte Address Offset (AD 17–0) | Module | Register | Access | Write by | Reference |
|---|---|---|---|---|---|
| 0x3_1008 | Controller | Interrupt enable | R/W | byte[1] | 10.5.4.2/10-50 |
| 0x3_1010 | | Interrupt status | R | — | 10.5.4.2.2/10-53 |
| 0x3_1018 | | Interrupt clear | R/W | byte | 10.5.4.3/10-54 |
| 0x3_1020 | | Identification | R | — | 10.5.4.4/10-54 |
| 0x3_1028 | | EU assignment status | R | — | 10.5.4.1/10-50 |
| 0x3_1030 | | Master control | R/W | byte | 10.5.4.6/10-55 |
| 0x3_1108 | Channel_1 | Configuration register | R/W | word | 10.4.4.1/10-37 |
| 0x3_1110 | | Pointer status | R/W | word | 10.4.4.2/10-41 |
| 0x3_1140 | | Current descriptor pointer | R | — | 10.4.4.3/10-43 |
| 0x3_1148 | | Fetch FIFO | W | word | 10.4.4.4/10-44 |
| 0x3_1180–0x3_11BF | | Descriptor buffer | R | — | 10.4.5.1/10-45 |
| 0x3_11C0–0x3_11DF | | Gather Link Table | R | — | 10.4.5.2/10-45 |
| 0x3_11E0–0x3_11FF | | Scatter Link Table | R | — | 10.4.5.2/10-45 |
| 0x3_1208 | Channel_2 | Configuration register | R/W | word | 10.4.4.1/10-37 |
| 0x3_1210 | | Pointer status | R/W | word | 10.4.4.2/10-41 |
| 0x3_1240 | | Current descriptor pointer | R | — | 10.4.4.3/10-43 |
| 0x3_1248 | | Fetch FIFO | W | word | 10.4.4.4/10-44 |
| 0x3_1280–0x3_12BF | | Descriptor buffer | R | — | 10.4.5.1/10-45 |
| 0x3_12C0–0x3_12DF | | Gather Link Table | R | — | 10.4.5.2/10-45 |
| 0x3_12E0–0x3_12FF | | Scatter Link Table | R | — | 10.4.5.2/10-45 |
| 0x3_1308 | Channel_3 | Configuration register | R/W | word | 10.4.4.1/10-37 |
| 0x3_1310 | | Pointer status | R/W | word | 10.4.4.2/10-41 |
| 0x3_1340 | | Current descriptor pointer | R | — | 10.4.4.3/10-43 |
| 0x3_1348 | | Fetch FIFO | W | word | 10.4.4.4/10-44 |
| 0x3_1380–0x3_13BF | | Descriptor buffer | R | — | 10.4.5.1/10-45 |
| 0x3_13C0–0x3_13DF | | Gather Link Table | R | — | 10.4.5.2/10-45 |
| 0x3_13E0–0x3_13FF | | Scatter Link Table | R | — | 10.4.5.2/10-45 |

Table 1-4. SEC Address Map (continued)

| Byte Address Offset (AD 17–0) | Module | Register | Access | Write by | Reference |
|---|---|---|---|---|---|
| 0x3_1408 | Channel_4 | Configuration register | R/W | word | 10.4.4.1/10-37 |
| 0x3_1410 | | Pointer status | R/W | word | 10.4.4.2/10-41 |
| 0x3_1440 | | Current descriptor pointer | R | — | 10.4.4.3/10-43 |
| 0x3_1448 | | Fetch FIFO | W | word | 10.4.4.4/10-44 |
| 0x3_1480–0x3_14BF | | Descriptor buffer | R | — | 10.4.5.1/10-45 |
| 0x3_14C0–0x3_14DF | | Gather Link Table | R | — | 10.4.5.2/10-45 |
| 0x3_14E0–0x3_14FF | | Scatter Link Table | R | — | 10.4.5.2/10-45 |
| 0x3_1500 | Poly-Channel | Fetch FIFO Enqueue Count | R/W | word | 10.4.3.1.1/10-35 |
| 0x3_1508 | | Descriptor Finished Count | R/W | word | 10.4.3.1.2/10-36 |
| 0x3_1510 | | Data Bytes In Count | R/W | word | 10.4.3.1.3/10-36 |
| 0x3_1518 | | Data Bytes Out Count | R/W | word | 10.4.3.1.4/10-37 |
| 0x3_1BF8 | Controller | IP block revision | R | — | 10.5.4.5/10-54 |
| 0x3_2000 | DEU | Mode register | R/W | word | 10.7.4.1/10-109 |
| 0x3_2008 | | Key size register | R/W | word | 10.7.4.2/10-110 |
| 0x3_2010 | | Data size register | R/W | word | 10.7.4.3/10-110 |
| 0x3_2018 | | Reset control register | R/W | word | 10.7.4.4/10-111 |
| 0x3_2028 | | Status register | R | — | 10.7.4.5/10-112 |
| 0x3_2030 | | Interrupt status register | R/W | word | 10.7.4.6/10-113 |
| 0x3_2038 | | Interrupt mask register | R/W | word | 10.7.4.7/10-115 |
| 0x3_2050 | | EU-Go | W | word | 10.7.4.8/10-116 |
| 0x3_2100 | | IV register | R/W | word | 10.7.4.9/10-117 |
| 0x3_2400 | | Key 1 register | W | byte | 10.7.4.10/10-117 |
| 0x3_2408 | | Key 2 register | W | byte | 10.7.4.10/10-117 |
| 0x3_2410 | | Key 3 register | W | byte | 10.7.4.10/10-117 |
| 0x3_2800–0x3_2FFF | | Input FIFO / Output FIFO | R/W$^2$ | byte | 10.7.4.11/10-117 |

Table 1-4. SEC Address Map (continued)

| Byte Address Offset (AD 17–0) | Module | Register | Access | Write by | Reference |
|---|---|---|---|---|---|
| 0x3_4000 | AESU | Mode register | R/W | word | 10.7.1.2/10-58 |
| 0x3_4008 | | Key size register | R/W | word | 10.7.1.3/10-61 |
| 0x3_4010 | | Data size register | R/W | word | 10.7.1.4/10-61 |
| 0x3_4018 | | Reset control register | R/W | word | 10.7.1.5/10-62 |
| 0x3_4028 | | Status register | R | — | 10.7.1.6/10-62 |
| 0x3_4030 | | Interrupt status register | R/W | word | 10.7.1.7/10-64 |
| 0x3_4038 | | Interrupt mask register | R/W | word | 10.7.1.8/10-66 |
| 0x3_4040 | | ICV size register | R/W | word | 10.7.1.9/10-67 |
| 0x3_4050 | | End of message register | W | word | 10.7.1.10/10-68 |
| 0x3_4100–0x3_415F | | Context | R/W | byte | 10.7.1.11/10-68 |
| 0x3_4400–0x3_441F | | Key registers | R/W | byte | 10.7.1.12/10-87 |
| 0x3_4800–0x3_4FFF | | Input FIFO / Output FIFO | R/W [1] | byte | 10.7.1.12.1/10-88 |
| 0x3_6000 | MDEU | Mode register | R/W | word | 10.7.6.2/10-132 |
| 0x3_6008 | | Key size register | R/W | word | 10.7.6.4/10-136 |
| 0x3_6010 | | Data size register | R/W | word | 10.7.6.5/10-137 |
| 0x3_6018 | | Reset control register | R/W | word | 10.7.6.6/10-137 |
| 0x3_6028 | | Status register | R | — | 10.7.6.7/10-138 |
| 0x3_6030 | | Interrupt status register | R/W | word | 10.7.6.8/10-139 |
| 0x3_6038 | | Interrupt mask register | R/W | word | 10.7.6.9/10-141 |
| 0x3_6040 | | ICV size register | W | word | 10.7.6.10/10-142 |
| 0x3_6050 | | End of message reigster | W | word | 10.7.6.11/10-143 |
| 0x3_6100–0x3_6147 | | Context registers | R/W | byte | 10.7.6.12/10-143 |
| 0x3_6400–0x3_647F | | Key registers | W | byte | 10.7.6.13/10-146 |
| 0x3_6800–0x3_6FFF | | Input FIFO | W [1] | byte | 10.7.6.14/10-146 |

Table 1-4. SEC Address Map (continued)

| Byte Address Offset (AD 17–0) | Module | Register | Access | Write by | Reference |
|---|---|---|---|---|---|
| 0x3_8000 | AFEU | Mode register | R/W | word | 10.7.2.1/10-89 |
| 0x3_8008 | | Key size register | R/W | word | 10.7.2.2/10-89 |
| 0x3_8010 | | Data size register | R/W | word | 10.7.2.3/10-90 |
| 0x3_8018 | | Reset control register | R/W | word | 10.7.2.4/10-91 |
| 0x3_8028 | | Status register | R | — | 10.7.2.5/10-91 |
| 0x3_8030 | | Interrupt status register | R/W | word | 10.7.2.6/10-92 |
| 0x3_8038 | | Interrupt mask register | R/W | word | 10.7.2.7/10-94 |
| 0x3_8050 | | End of message register | W | word | 10.7.2.8/10-96 |
| 0x3_8100–0x3_81FF | | Context memory | R/W | byte | 10.7.2.9/10-96 |
| 0x3_8200 | | Context memory pointers | R/W | byte | 10.7.2.9/10-96 |
| 0x3_8400–0x3_840F | | Key registers | W | byte | 10.7.2.10/10-97 |
| 0x3_8800–0x3_8FFF<br>(3_8E00) | | Input FIFO / Output FIFO<br>(special context address) | R/W [1] | byte | 10.7.2.10.1/10-97 |
| 0x3_A000 | RNGU | Mode register | R/W | word | 10.7.8.1/10-156 |
| 0x3_A010 | | Data size register | R/W | word | 10.7.8.2/10-156 |
| 0x3_A018 | | Reset control register | R/W | word | 10.7.8.3/10-156 |
| 0x3_A028 | | Status register | R | — | 10.7.8.4/10-157 |
| 0x3_A030 | | Interrupt status register | R/W | word | 10.7.8.5/10-158 |
| 0x3_A038 | | Interrupt mask register | R/W | word | 10.7.8.6/10-159 |
| 0x3_A050 | | End of message register | W | word | 10.7.8.7/10-160 |
| 0x3_A400–0x3_A43F | | Entropy registers | W | word | 10.7.8.8/10-161 |
| 0x3_A800–0x3_AFFF | | Output FIFO | R [1] | — | 10.7.8.8/10-161 |

Table 1-4. SEC Address Map (continued)

| Byte Address Offset (AD 17–0) | Module | Register | Access | Write by | Reference |
|---|---|---|---|---|---|
| 0x3_C000 | PKEU | Mode register | R/W | word | 10.7.7.1/10-147 |
| 0x3_C008 | | Key size register | R/W | word | 10.7.7.2/10-147 |
| 0x3_C010 | | Data size register | R/W | word | 10.7.7.4/10-149 |
| 0x3_C018 | | Reset control register | R/W | word | 10.7.7.5/10-149 |
| 0x3_C028 | | Status register | R | — | 10.7.7.6/10-150 |
| 0x3_C030 | | Interrupt status register | R/W | word | 10.7.7.7/10-151 |
| 0x3_C038 | | Interrupt mask register | R/W | word | 10.7.7.8/10-153 |
| 0x3_C040 | | ABSize | R/W | word | 10.7.7.3/10-148 |
| 0x3_C050 | | End of message register | W | word | 10.7.7.9/10-154 |
| 0x3_C200–0x3_C27F | | Parameter memory A0 | R/W | byte | 10.7.7.10/10-154 |
| 0x3_C280–0x3_C2FF | | Parameter memory A1 | R/W | byte | |
| 0x3_C300–0x3_C37F | | Parameter memory A2 | R/W | byte | |
| 0x3_C380–0x3_C3FF | | Parameter memory A3 | R/W | byte | |
| 0x3_C400–0x3_C47F | | Parameter memory B0 | R/W | byte | |
| 0x3_C480–0x3_C4FF | | Parameter memory B1 | R/W | byte | |
| 0x3_C500–0x3_C57F | | Parameter memory B2 | R/W | byte | |
| 0x3_C580–0x3_C5FF | | Parameter memory B3 | R/W | byte | |
| 0x3_C800–0x3_C9FF | | Parameter memory N | R/W | byte | |
| 0x3_CA00–0x3_CBFF | | Parameter memory E | W | byte | |

Table 1-4. SEC Address Map (continued)

| Byte Address Offset (AD 17–0) | Module | Register | Access | Write by | Reference |
|---|---|---|---|---|---|
| 0x3_E000 | KEU | Mode register | R/W | word | 10.7.5.1/10-118 |
| 0x3_E008 | | Key size register | R/W | word | 10.7.5.2/10-119 |
| 0x3_E010 | | Data size register | R/W | word | 10.7.5.3/10-120 |
| 0x3_E018 | | Reset control register | R/W | word | 10.7.5.4/10-121 |
| 0x3_E028 | | Status register | R | — | 10.7.5.5/10-122 |
| 0x3_E030 | | Interrupt Status register | R/W | word | 10.7.5.6/10-123 |
| 0x3_E038 | | Interrupt Mask register | R/W | word | 10.7.5.7/10-125 |
| 0x3_E048 | | Data out register (f9 MAC) | R | — | 10.7.5.8/10-127 |
| 0x3_E050 | | End of message register | W | word | 10.7.5.9/10-127 |
| 0x3_E100 | | IV_1 register | R/W | byte | 10.7.5.10/10-128 |
| 0x3_E108 | | ICV_In register | R/W | byte | 10.7.5.11/10-129 |
| 0x3_E110 | | IV_2 register (FRESH) | R/W | byte | 10.7.5.12/10-129 |
| 0x3_E118 | | Context_1 register | R/W | byte | 10.7.5.13/10-129 |
| 0x3_E120 | | Context_2 register | R/W | byte | 10.7.5.13/10-129 |
| 0x3_E128 | | Context_3 register | R/W | byte | 10.7.5.13/10-129 |
| 0x3_E130 | | Context_4 register | R/W | byte | 10.7.5.13/10-129 |
| 0x3_E138 | | Context_5 register | R/W | byte | 10.7.5.13/10-129 |
| 0x3_E140 | | Context_6 register | R/W | byte | 10.7.5.13/10-129 |
| 0x3_E400 | | Key data register_1 (CK-high) | R/W | byte | 10.7.5.14/10-130 |
| 0x3_E408 | | Key data register_2 (CK-low) | R/W | byte | 10.7.5.14/10-130 |
| 0x3_E410 | | Key dataregister_3 (IK-high) | R/W | byte | 10.7.5.15/10-130 |
| 0x3_E418 | | Key data register_4 (IK-low) | R/W | byte | 10.7.5.15/10-130 |
| 0x3_E800–0x3_EFFF | | Input FIFO / Output FIFO | R/W [1] | byte | 10.7.5.16/10-131 |

Table 1-4. SEC Address Map (continued)

| Byte Address Offset (AD 17–0) | Module | Register | Access | Write by | Reference |
|---|---|---|---|---|---|
| 0x3_F000 | CRCU | Mode register | R/W | word | 10.7.3.2/10-98 |
| 0x3_F008 | | Key size register | R/W | word | 10.7.3.3/10-99 |
| 0x3_F010 | | Data size register | R/W | word | 10.7.3.4/10-100 |
| 0x3_F018 | | Reset control register | R/W | word | 10.7.3.5/10-100 |
| 0x3_F020 | | Control | R/W | word | 10.7.3.6/10-101 |
| 0x3_F028 | | Status register | R | — | 10.7.3.7/10-101 |
| 0x3_F030 | | Interrupt status register | R/W | word | 10.7.3.8/10-102 |
| 0x3_F038 | | Interrupt mask register | R/W | word | 10.7.3.9/10-104 |
| 0x3_F040 | | ICV size register | R/W | word | 10.7.1.9/10-67 |
| 0x3_F050 | | End of message register | W | word | 10.7.3.11/10-106 |
| 0x3_F108 | | Context register | R/W | byte | 10.7.3.12/10-106 |
| 0x3_F400 | | Key register | R/W | byte | 10.7.3.13/10-108 |
| 0x3_F800–0x3_FFFF | | Input FIFO | W [1] | byte | 10.7.3.14/10-108 |

[1] Byte accessibility is controlled by internal logic, particularly at FIFOs, to prevent unintended overwrites of partial words during writes, and to prevent unintended duplicate reads of partial data during reads. In addition, these bytes must be presented on the correct byte lanes for the intended destination.

[2] For the EU FIFOs, write operations anywhere in the address range enqueue to the input FIFO, and read operations anywhere in the address range dequeue from the output FIFO. See the referenced section for more detailed information.

## A.2 Programmable Interrupt Controller (PIC)

The programmable interrupt controller (PIC) follows the OpenPIC programming model which requires a larger register address space than the 4 Kbytes allocated to other blocks within the general utilities space. For this reason, the PIC is allocated the second 256 Kbytes of CCSR space (0x4_0000–0x6_FFFF).

### A.2.1 PIC—Global Registers

Table A-14. PIC Global Registers

| PIC Global Registers—Block Base Address 0x4_0000 | | | | |
|---|---|---|---|---|
| Offset | Register | Access | Reset | Section/Page |
| 0x0000 | BRR1—Block revision register 1 | R | 0x0040_0300 | 9.3.1.1/9-19 |
| 0x0010 | BRR2—Block revision register 2 | R | 0x0000_0001 | 9.3.1.2/9-19 |
| 0x0020–0x0030 | Reserved | — | — | — |

## Table A-14. PIC Global Registers (continued)

| PIC Global Registers—Block Base Address 0x4_0000 | | | | |
|---|---|---|---|---|
| **Offset** | **Register** | **Access** | **Reset** | **Section/Page** |
| 0x0040 | IPIDR0—Interprocessor interrupt 0 (IPI 0) dispatch register | W | 0x0000_0000 | 9.3.8.1/9-48 |
| 0x0050 | IPIDR1—IPI 1 dispatch register | | | |
| 0x0060 | IPIDR2—IPI 2 dispatch register | | | |
| 0x0070 | IPIDR3—IPI 3 dispatch register | | | |
| 0x0080 | CTPR—Current task priority register | R/W | 0x0000_000F | 9.3.8.2/9-49 |
| 0x0090 | WHOAMI—Who am I register | R | n/a | 9.3.8.3/9-50 |
| 0x00A0 | IACK—Interrupt acknowledge register | R | 0x0000_0000 | 9.3.8.4/9-50 |
| 0x00B0 | EOI—End of interrupt register | W | 0x0000_0000 | 9.3.8.5/9-51 |
| 0x00C0–0x0FF0 | Reserved | — | — | — |
| 0x1000 | FRR—Feature reporting register | R | 0x006B_0*n*020 x0067_0002 | 9.3.1.3/9-20 |
| 0x1010 | Reserved | — | — | — |
| 0x1020 | GCR—Global configuration register | R/W | 0x0000_0000 | 9.3.1.4/9-21 |
| 0x1030 | Reserved | — | — | — |
| 0x1040–0x1070 | Vendor reserved | — | — | — |
| 0x1080 | VIR—Vendor identification register | R | 0x0000_0000 | 9.3.1.5/9-21 |
| 0x1090 | PIR—Processor core initialization register | R/W | 0x0000_0000 | 9.3.1.6/9-22 |
| 0x10A0 | IPIVPR0—IPI 0 vector/priority register | R/W | 0x8000_0000 | 9.3.1.7/9-22 |
| 0x10B0 | IPIVPR1—IPI 1 vector/priority register | | | |
| 0x10C0 | IPIVPR2—IPI 2 vector/priority register | | | |
| 0x10D0 | IPIVPR3—IPI 3 vector/priority register | | | |
| 0x10E0 | SVR—Spurious vector register | R/W | 0x0000_FFFF | 9.3.1.8/9-23 |
| Global Timer Group A Registers | | | | |
| 0x10F0 | TFRRA—Timer frequency reporting register (Group A) | R/W | 0x0000_0000 | 9.3.2.1/9-24 |
| 0x1100 | GTCCRA0—Global timer 0 current count register (Group A) | R | 0x0000_0000 | 9.3.2.2/9-24 |
| 0x1110 | GTBCRA0—Global timer 0 base count register (Group A) | R/W | 0x8000_0000 | 9.3.2.3/9-25 |
| 0x1120 | GTVPRA0—Global timer 0 vector/priority register (Group A) | R/W | 0x8000_0000 | 9.3.2.4/9-25 |
| 0x1130 | GTDRA0—Global timer 0 destination register (Group A) | R/W | 0x0000_0001 | 9.3.2.5/9-26 |
| 0x1140 | GTCCRA1—Global timer 1 current count register (Group A) | R | 0x0000_0000 | 9.3.2.2/9-24 |
| 0x1150 | GTBCRA1—Global timer 1 base count register (Group A) | R/W | 0x8000_0000 | 9.3.2.3/9-25 |
| 0x1160 | GTVPRA1—Global timer 1 vector/priority register (Group A) | R/W | 0x8000_0000 | 9.3.2.4/9-25 |
| 0x1170 | GTDRA1—Global timer 1 destination register (Group A) | R/W | 0x0000_0001 | 9.3.2.5/9-26 |
| 0x1180 | GTCCRA2—Global timer 2 current count register (Group A) | R | 0x0000_0000 | 9.3.2.2/9-24 |
| 0x1190 | GTBCRA2—Global timer 2 base count register (Group A) | R/W | 0x8000_0000 | 9.3.2.3/9-25 |
| 0x11A0 | GTVPRA2—Global timer 2 vector/priority register (Group A) | R/W | 0x8000_0000 | 9.3.2.4/9-25 |

Table A-14. PIC Global Registers (continued)

| PIC Global Registers—Block Base Address 0x4_0000 | | | | |
|---|---|---|---|---|
| Offset | Register | Access | Reset | Section/Page |
| 0x11B0 | GTDRA2—Global timer 2 destination register (Group A) | R/W | 0x0000_0001 | 9.3.2.5/9-26 |
| 0x11C0 | GTCCRA3—Global timer 3 current count register (Group A) | R | 0x0000_0000 | 9.3.2.2/9-24 |
| 0x11D0 | GTBCRA3—Global timer 3 base count register (Group A) | R/W | 0x8000_0000 | 9.3.2.3/9-25 |
| 0x11E0 | GTVPRA3—Global timer 3 vector/priority register (Group A) | R/W | 0x8000_0000 | 9.3.2.4/9-25 |
| 0x11F0 | GTDRA3—Global timer 3 destination register (Group A) | R/W | 0x0000_0001 | 9.3.2.5/9-26 |
| 0x1200–0x12F0 | Reserved | — | — | — |
| 0x1300 | TCRA—Timer control register (Group A) | R/W | 0x0000_0000 | 9.3.2.6/9-27 |
| 0x1308 | ERQSR—External interrupt summary register | R | 0x0000_0000 | 9.3.3.1/9-29 |
| 0x1310 | IRQSR0—IRQ_OUT summary register 0 | R | 0x0000_0000 | 9.3.3.2/9-29 |
| 0x1320 | IRQSR1—IRQ_OUT summary register 1 | R | 0x0000_0000 | 9.3.3.3/9-30 |
| 0x1324 | IRQSR2—IRQ_OUT summary register 2 | R | 0x0000_0000 | 9.3.3.4/9-31 |
| 0x1330 | CISR0—Critical interrupt summary register 0 | R | 0x0000_0000 | 9.3.3.5/9-31 |
| 0x1340 | CISR1—Critical interrupt summary register 1 | R | 0x0000_0000 | 9.3.3.6/9-32 |
| 0x1344 | CISR2—Critical interrupt summary register 2 | R | 0x0000_0000 | 9.3.3.7/9-32 |
| 0x1350 | PM0MR0—Performance monitor 0 mask register 0 | R/W | 0xFFFF_FFFF | 9.3.4.1/9-33 |
| 0x1360 | PM0MR1—Performance monitor 0 mask register 1 | R/W | 0xFFFF_FFFF | 9.3.4.2/9-34 |
| 0x1364 | PM0MR2—Performance monitor 0 mask register 2 | R/W | 0xFFFF_FFFF | 9.3.4.2/9-34 |
| 0x1370 | PM1MR0—Performance monitor 1 mask register 0 | R/W | 0xFFFF_FFFF | 9.3.4.1/9-33 |
| 0x1380 | PM1MR1—Performance monitor 1 mask register 1 | R/W | 0xFFFF_FFFF | 9.3.4.2/9-34 |
| 0x1384 | PM1MR2—Performance monitor 1 mask register 2 | R/W | 0xFFFF_FFFF | 9.3.4.2/9-34 |
| 0x1390 | PM2MR0—Performance monitor 2 mask register 0 | R/W | 0xFFFF_FFFF | 9.3.4.1/9-33 |
| 0x13A0 | PM2MR1—Performance monitor 2 mask register 1 | R/W | 0xFFFF_FFFF | 9.3.4.2/9-34 |
| 0x13A4 | PM2MR2—Performance monitor 2 mask register 2 | R/W | 0xFFFF_FFFF | 9.3.4.2/9-34 |
| 0x13B0 | PM3MR0—Performance monitor 3 mask register 0 | R/W | 0xFFFF_FFFF | 9.3.4.1/9-33 |
| 0x13C0 | PM3MR1—Performance monitor 3 mask register 1 | R/W | 0xFFFF_FFFF | 9.3.4.2/9-34 |
| 0x13C4 | PM3MR2—Performance monitor 3 mask register 2 | R/W | 0xFFFF_FFFF | 9.3.4.2/9-34 |
| 0x13D0–0x13F0 | Reserved | — | — | — |
| 0x1400 | MSGR0—Message register 0 | R/W | 0x0000_0000 | 9.3.5.1/9-35 |
| 0x1410 | MSGR1—Message register 1 | R/W | 0x0000_0000 | 9.3.5.1/9-35 |
| 0x1420 | MSGR2—Message register 2 | R/W | 0x0000_0000 | 9.3.5.1/9-35 |
| 0x1430 | MSGR3—Message register 3 | R/W | 0x0000_0000 | 9.3.5.1/9-35 |
| 0x1440–0x14F0 | Reserved | — | — | — |
| 0x1500 | MER—Message enable register | R/W | 0x0000_0000 | 9.3.5.2/9-35 |
| 0x1510 | MSR—Message status register | R/W | 0x0000_0000 | 9.3.5.3/9-36 |

### Table A-14. PIC Global Registers (continued)

| PIC Global Registers—Block Base Address 0x4_0000 | | | | |
|---|---|---|---|---|
| **Offset** | **Register** | **Access** | **Reset** | **Section/Page** |
| 0x1520–<br>0x15F0 | Reserved | — | — | — |
| 0x1600 | MSIR0—Shared message signaled interrupt register 0 | RC | 0x0000_0000 | 9.3.6.1/9-37 |
| 0x1610 | MSIR1—Shared message signaled interrupt register 1 | RC | 0x0000_0000 | 9.3.6.1/9-37 |
| 0x1620 | MSIR2—Shared message signaled interrupt register 2 | RC | 0x0000_0000 | 9.3.6.1/9-37 |
| 0x1630 | MSIR3—Shared message signaled interrupt register 3 | RC | 0x0000_0000 | 9.3.6.1/9-37 |
| 0x1640 | MSIR4—Shared message signaled interrupt register 4 | RC | 0x0000_0000 | 9.3.6.1/9-37 |
| 0x1650 | MSIR5—Shared message signaled interrupt register 5 | RC | 0x0000_0000 | 9.3.6.1/9-37 |
| 0x1660 | MSIR6—Shared message signaled interrupt register 6 | RC | 0x0000_0000 | 9.3.6.1/9-37 |
| 0x1670 | MSIR7—Shared message signaled interrupt register 7 | RC | 0x0000_0000 | 9.3.6.1/9-37 |
| 0x1680–<br>0x1700 | Reserved | — | — | — |
| 0x1720 | MSISR—Shared message signaled interrupt status register | R | 0x0000_0000 | 9.3.6.2/9-37 |
| 0x1740 | MSIIR—Shared message signaled interrupt index register | W | 0x0000_0000 | 9.3.6.3/9-38 |
| 0x1750–<br>0x20E0 | Reserved | — | — | — |
| Global Timer Group B Registers | | | | |
| 0x20F0 | TFRRB—Timer frequency reporting register group B | R/W | 0x0000_0000 | 9.3.2.1/9-24 |
| 0x2100 | GTCCRB0—Global timer current count register group B 0 | R | 0x0000_0000 | 9.3.2.2/9-24 |
| 0x2110 | GTBCRB0—Global timer base count register group B 0 | R/W | 0x8000_0000 | 9.3.2.3/9-25 |
| 0x2120 | GTVPRB0—Global timer vector/priority register group B 0 | R/W | 0x8000_0000 | 9.3.2.4/9-25 |
| 0x2130 | GTDRB0—Global timer destination register group B 0 | R/W | 0x0000_0001 | 9.3.2.5/9-26 |
| 0x2140 | GTCCRB1—Global timer current count register group B 1 | R | 0x0000_0000 | 9.3.2.2/9-24 |
| 0x2150 | GTBCRB1—Global timer base count register group B 1 | R/W | 0x8000_0000 | 9.3.2.3/9-25 |
| 0x2160 | GTVPRB1—Global timer vector/priority register group B 1 | R/W | 0x8000_0000 | 9.3.2.4/9-25 |
| 0x2170 | GTDRB1—Global timer destination register group B 1 | R/W | 0x0000_0001 | 9.3.2.5/9-26 |
| 0x2180 | GTCCRB2—Global timer current count register group B 2 | R | 0x0000_0000 | 9.3.2.2/9-24 |
| 0x2190 | GTBCRB2—Global timer base count register group B 2 | R/W | 0x8000_0000 | 9.3.2.3/9-25 |
| 0x21A0 | GTVPRB2—Global timer vector/priority register group B 2 | R/W | 0x8000_0000 | 9.3.2.4/9-25 |
| 0x21B0 | GTDRB2—Global timer destination register group B 2 | R/W | 0x0000_0001 | 9.3.2.5/9-26 |
| 0x21C0 | GTCCRB3—Global timer current count register group B 3 | R | 0x0000_0000 | 9.3.2.2/9-24 |
| 0x21D0 | GTBCRB3—Global timer base count register group B 3 | R/W | 0x8000_0000 | 9.3.2.3/9-25 |
| 0x21E0 | GTVPRB3—Global timer vector/priority register group B 3 | R/W | 0x8000_0000 | 9.3.2.4/9-25 |
| 0x21F0 | GTDRB3—Global timer destination register group B 3 | R/W | 0x0000_0001 | 9.3.2.5/9-26 |
| 0x2200–<br>0x22F0 | Reserved | — | — | — |
| 0x2300 | TCRB—Timer control register (Group B) | R/W | 0x0000_0000 | 9.3.2.6/9-27 |

Table A-14. PIC Global Registers (continued)

| PIC Global Registers—Block Base Address 0x4_0000 | | | | |
|---|---|---|---|---|
| **Offset** | **Register** | **Access** | **Reset** | **Section/Page** |
| 0x2310–<br>0x23F0 | Reserved | — | — | — |
| 0x2400 | MSGR4—Message register 4 | R/W | 0x0000_0000 | 9.3.5.1/9-35 |
| 0x2410 | MSGR5—Message register 5 | | | |
| 0x2420 | MSGR6—Message register 6 | | | |
| 0x2430 | MSGR7—Message register 7 | | | |
| 0x2440–<br>0x24F0 | Reserved | — | — | — |
| 0x2500 | MER—Message enable register (for MSGR4–7) | R/W | 0x0000_0000 | 9.3.5.2/9-35 |
| 0x2510 | MSR—Message status register (for MSGG4–7) | R/W | 0x0000_0000 | 9.3.5.3/9-36 |
| 0x2514–<br>0xFFF0 | Reserved | — | — | — |

## A.2.2 PIC—Interrupt Source Registers

Table A-15. PIC Interrupt Source Registers

| PIC Interrupt Source Registers—Block Base Address 0x5_0000 | | | | |
|---|---|---|---|---|
| **Offset** | **Register** | **Access** | **Reset** | **Section/Page** |
| 0x0000 | EIVPR0—External interrupt 0 (IRQ0) vector/priority register or PEX1-INTA vector/priority register | R/W | 0x8000_0000 | 9.3.7.1/9-41 |
| 0x0010 | EIDR0—External interrupt 0 (IRQ0) destination register or PEX1-INTA destination register | R/W | 0x0000_0001 | 9.3.7.2/9-42 |
| 0x0020 | EIVPR1—External interrupt 1 (IRQ1) vector/priority register or PEX1-INTB vector/priority register | R/W | 0x8000_0000 | 9.3.7.1/9-41 |
| 0x0030 | EIDR1—External interrupt 1 (IRQ1) destination register or PEX1-INTB destination register | R/W | 0x0000_0001 | 9.3.7.2/9-42 |
| 0x0040 | EIVPR2—External interrupt 2 (IRQ2) vector/priority register or PEX1-INTC vector/priority register | R/W | 0x8000_0000 | 9.3.7.1/9-41 |
| 0x0050 | EIDR2—External interrupt 2 (IRQ2) destination register or PEX1-INTC destination register | R/W | 0x0000_0001 | 9.3.7.2/9-42 |
| 0x0060 | EIVPR3—External interrupt 3 (IRQ3) vector/priority register or PEX1-INTD vector/priority register | R/W | 0x8000_0000 | 9.3.7.1/9-41 |
| 0x0070 | EIDR3—External interrupt 3 (IRQ3) destination register or PEX1-INTD destination register | R/W | 0x0000_0001 | 9.3.7.2/9-42 |
| 0x0080 | EIVPR4—External interrupt 4 (IRQ4) vector/priority register or PEX2-INTA vector/priority register | R/W | 0x8000_0000 | 9.3.7.1/9-41 |
| 0x0090 | EIDR4—External interrupt 4 (IRQ4) destination register or PEX2-INTA destination register | R/W | 0x0000_0001 | 9.3.7.2/9-42 |
| 0x00A0 | EIVPR5—External interrupt 5 (IRQ5) vector/priority register or PEX2-INTB vector/priority register | R/W | 0x8000_0000 | 9.3.7.1/9-41 |

Table A-15. PIC Interrupt Source Registers (continued)

| PIC Interrupt Source Registers—Block Base Address 0x5_0000 | | | | |
|---|---|---|---|---|
| Offset | Register | Access | Reset | Section/Page |
| 0x00B0 | EIDR5—External interrupt 5 (IRQ5) destination register or PEX2-INTB destination register | R/W | 0x0000_0001 | 9.3.7.2/9-42 |
| 0x00C0 | EIVPR6—External interrupt 6 (IRQ6) vector/priority register or PEX2-INTC vector/priority register | R/W | 0x8000_0000 | 9.3.7.1/9-41 |
| 0x00D0 | EIDR6—External interrupt 6 (IRQ6) destination register or PEX2-INTC destination register | R/W | 0x0000_0001 | 9.3.7.2/9-42 |
| 0x00E0 | EIVPR7—External interrupt 7 (IRQ7) vector/priority register or PEX2-INTD vector/priority register | R/W | 0x8000_0000 | 9.3.7.1/9-41 |
| 0x00F0 | EIDR7—External interrupt 7 (IRQ7) destination register or PEX2-INTD destination register | R/W | 0x0000_0001 | 9.3.7.2/9-42 |
| 0x0100 | EIVPR8—External interrupt 8 (IRQ8) vector/priority register or PEX3-INTA vector/priority register | R/W | 0x8000_0000 | 9.3.7.1/9-41 |
| 0x0110 | EIDR8—External interrupt 8 (IRQ8) destination register or PEX3-INTA destination register | R/W | 0x0000_0001 | 9.3.7.2/9-42 |
| 0x0120 | EIVPR9—External interrupt 9 (IRQ9) vector/priority register or PEX3-INTB vector/priority register | R/W | 0x8000_0000 | 9.3.7.1/9-41 |
| 0x0130 | EIDR9—External interrupt 9 (IRQ9) destination register or PEX3-INTB destination register | R/W | 0x0000_0001 | 9.3.7.2/9-42 |
| 0x0140 | EIVPR10—External interrupt 10 (IRQ10) vector/priority register<br>or PEX3-INTC vector/priority register | R/W | 0x8000_0000 | 9.3.7.1/9-41 |
| 0x0150 | EIDR10—External interrupt 10 (IRQ10) destination register or PEX3-INTC destination register | R/W | 0x0000_0001 | 9.3.7.2/9-42 |
| 0x0160 | EIVPR11—External interrupt 11 (IRQ11) vector/priority register<br>or PEX3-INTD vector/priority register | R/W | 0x8000_0000 | 9.3.7.1/9-41 |
| 0x0170 | EIDR11—External interrupt 11 (IRQ11) destination register or PEX3-INTD destination register | R/W | 0x0000_0001 | 9.3.7.2/9-42 |
| 0x0180–0x01F0 | Reserved | — | — | — |
| 0x0200 | IIVPR0—Internal interrupt 0 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0210 | IIDR0—Internal interrupt 0 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0220 | IIVPR1—Internal interrupt 1 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0230 | IIDR1—Internal interrupt 1 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0240 | IIVPR2—Internal interrupt 2 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0250 | IIDR2—Internal interrupt 2 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0260 | IIVPR3—Internal interrupt 3 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0270 | IIDR3—Internal interrupt 3 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0280 | IIVPR4—Internal interrupt 4 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0290 | IIDR4—Internal interrupt 4 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x02A0 | IIVPR5—Internal interrupt 5 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |

### Table A-15. PIC Interrupt Source Registers (continued)

| PIC Interrupt Source Registers—Block Base Address 0x5_0000 | | | | |
|---|---|---|---|---|
| **Offset** | **Register** | **Access** | **Reset** | **Section/Page** |
| 0x02B0 | IIDR5—Internal interrupt 5 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x02C0 | IIVPR6—Internal interrupt 6 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x02D0 | IIDR6—Internal interrupt 6 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x02E0 | IIVPR7—Internal interrupt 7 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x02F0 | IIDR7—Internal interrupt 7 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0300 | IIVPR8—Internal interrupt 8 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0310 | IIDR8—Internal interrupt 8 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0320 | IIVPR9—Internal interrupt 9 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0330 | IIDR9—Internal interrupt 9 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0340 | IIVPR10—Internal interrupt 10 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0350 | IIDR10—Internal interrupt 10 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0360 | IIVPR11—Internal interrupt 11 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0370 | IIDR11—Internal interrupt 11 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0380 | IIVPR12—Internal interrupt 12 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0390 | IIDR12—Internal interrupt 12 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x03A0 | IIVPR13—Internal interrupt 13 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x03B0 | IIDR13—Internal interrupt 13 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x03C0 | IIVPR14—Internal interrupt 14 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x03D0 | IIDR14—Internal interrupt 14 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x03E0 | IIVPR15—Internal interrupt 15 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x03F0 | IIDR15—Internal interrupt 15 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0400 | IIVPR16—Internal interrupt 16 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0410 | IIDR16—Internal interrupt 16 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0420 | IIVPR17—Internal interrupt 17 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0430 | IIDR17—Internal interrupt 17 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0440 | IIVPR18—Internal interrupt 18 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0450 | IIDR18—Internal interrupt 18 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0460 | IIVPR19—Internal interrupt 19 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0470 | IIDR19—Internal interrupt 19 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0480 | IIVPR20—Internal interrupt 20 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0490 | IIDR20—Internal interrupt 20 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x04A0 | IIVPR21—Internal interrupt 21 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x04B0 | IIDR21—Internal interrupt 21 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x04C0 | IIVPR22—Internal interrupt 22 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x04D0 | IIDR22—Internal interrupt 22 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x04E0 | IIVPR23—Internal interrupt 23 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |

### Table A-15. PIC Interrupt Source Registers (continued)

| PIC Interrupt Source Registers—Block Base Address 0x5_0000 | | | | |
|---|---|---|---|---|
| **Offset** | **Register** | **Access** | **Reset** | **Section/Page** |
| 0x04F0 | IIDR23—Internal interrupt 23 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0500 | IIVPR24—Internal interrupt 24 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0510 | IIDR24—Internal interrupt 24 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0520 | IIVPR25—Internal interrupt 25 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0530 | IIDR25—Internal interrupt 25 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0540 | IIVPR26—Internal interrupt 26 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0550 | IIDR26—Internal interrupt 26 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0560 | IIVPR27—Internal interrupt 27 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0570 | IIDR27—Internal interrupt 27 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0580 | IIVPR28—Internal interrupt 28 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0590 | IIDR28—Internal interrupt 28 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x05A0 | IIVPR29—Internal interrupt 29 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x05B0 | IIDR29—Internal interrupt 29 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x05C0 | IIVPR30—Internal interrupt 30 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x05D0 | IIDR30—Internal interrupt 30 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x05E0 | IIVPR31—Internal interrupt 31 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x05F0 | IIDR31—Internal interrupt 31 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0600 | IIVPR32—Internal interrupt 32 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0610 | IIDR32—Internal interrupt 32 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0620 | IIVPR33—Internal interrupt 33 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0630 | IIDR33—Internal interrupt 33 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0640 | IIVPR34—Internal interrupt 34 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0650 | IIDR34—Internal interrupt 34 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0660 | IIVPR35—Internal interrupt 35 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0670 | IIDR35—Internal interrupt 35 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0680 | IIVPR36—Internal interrupt 36 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0690 | IIDR36—Internal interrupt 36 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x06A0 | IIVPR37—Internal interrupt 37 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x06B0 | IIDR37—Internal interrupt 37 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x06C0 | IIVPR38—Internal interrupt 38 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x06D0 | IIDR38—Internal interrupt 38 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x06E0 | IIVPR39—Internal interrupt 39 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x06F0 | IIDR39—Internal interrupt 39 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0700 | IIVPR40—Internal interrupt 40 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0710 | IIDR40—Internal interrupt 40 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0720 | IIVPR41—Internal interrupt 41 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |

Table A-15. PIC Interrupt Source Registers (continued)

| PIC Interrupt Source Registers—Block Base Address 0x5_0000 | | | | |
|---|---|---|---|---|
| Offset | Register | Access | Reset | Section/Page |
| 0x0730 | IIDR41—Internal interrupt 41 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0740 | IIVPR42—Internal interrupt 42 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0750 | IIDR42—Internal interrupt 42 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0760 | IIVPR43—Internal interrupt 43 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0770 | IIDR43—Internal interrupt 43 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0780 | IIVPR44—Internal interrupt 44 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0790 | IIDR44—Internal interrupt 44 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x07A0 | IIVPR45—Internal interrupt 45 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x07B0 | IIDR45—Internal interrupt 45 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x07C0 | IIVPR46—Internal interrupt 46 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x07D0 | IIDR46—Internal interrupt 46 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x07E0 | IIVPR47—Internal interrupt 47 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x07F0 | IIDR47—Internal interrupt 47 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0800 | IIVPR48—Internal interrupt 48 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0810 | IIDR48—Internal interrupt 48 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0820 | IIVPR49—Internal interrupt 49 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0830 | IIDR49—Internal interrupt 49 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0840 | IIVPR50—Internal interrupt 50 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0850 | IIDR50—Internal interrupt 50 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0860 | IIVPR51—Internal interrupt 51 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0870 | IIDR51—Internal interrupt 51 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0880 | IIVPR52—Internal interrupt 52 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0890 | IIDR52—Internal interrupt 52 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x08A0 | IIVPR53—Internal interrupt 53 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x08B0 | IIDR53—Internal interrupt 53 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x08C0 | IIVPR54—Internal interrupt 54 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x08D0 | IIDR54—Internal interrupt 54 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x08E0 | IIVPR55—Internal interrupt 55 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x08F0 | IIDR55—Internal interrupt 55 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0900 | IIVPR56—Internal interrupt 56 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0910 | IIDR56—Internal interrupt 56 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0920 | IIVPR57—Internal interrupt 57 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0930 | IIDR57—Internal interrupt 57 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0940 | IIVPR58—Internal interrupt 58 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0950 | IIDR58—Internal interrupt 58 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0960 | IIVPR59—Internal interrupt 59 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |

Table A-15. PIC Interrupt Source Registers (continued)

| PIC Interrupt Source Registers—Block Base Address 0x5_0000 | | | | |
|---|---|---|---|---|
| Offset | Register | Access | Reset | Section/Page |
| 0x0970 | IIDR59—Internal interrupt 59 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x0980 | IIVPR60—Internal interrupt 60 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x0990 | IIDR60—Internal interrupt 60 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x09A0 | IIVPR61—Internal interrupt 61 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x09B0 | IIDR61—Internal interrupt 61 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x09C0 | IIVPR62—Internal interrupt 62 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x09D0 | IIDR62—Internal interrupt 62 destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x09E0 | IIVPR63—Internal interrupt 63 vector/priority register | R/W | 0x8080_0000 | 9.3.7.3/9-43 |
| 0x09F0 | IIDR63—Internal interrupt 63 destination register | R/W | 0x0000_0001 | 9.3.7.3/9-43 |
| 0x0A00–0x15F0 | Reserved | — | — | — |
| 0x1600 | MIVPR0—Messaging interrupt 0 (MSG 0) vector/priority register | R/W | 0x8000_0000 | 9.3.7.5/9-45 |
| 0x1610 | MIDR0—Messaging interrupt 0 (MSG 0) destination register | R/W | 0x0000_0001 | 9.3.7.6/9-46 |
| 0x1620 | MIVPR1—Messaging interrupt 1 (MSG 1) vector/priority register | R/W | 0x8000_0000 | 9.3.7.5/9-45 |
| 0x1630 | MIDR1—Messaging interrupt 1 (MSG 1) destination register | R/W | 0x0000_0001 | 9.3.7.6/9-46 |
| 0x1640 | MIVPR2—Messaging interrupt 2 (MSG 2) vector/priority register | R/W | 0x8000_0000 | 9.3.7.5/9-45 |
| 0x1650 | MIDR2—Messaging interrupt 2 (MSG 2) destination register | R/W | 0x0000_0001 | 9.3.7.6/9-46 |
| 0x1660 | MIVPR3—Messaging interrupt 3 (MSG 3) vector/priority register | R/W | 0x8000_0000 | 9.3.7.5/9-45 |
| 0x1670 | MIDR3—Messaging interrupt 3 (MSG 3) destination register | R/W | 0x0000_0001 | 9.3.7.4/9-44 |
| 0x1680 | MIVPR4—Messaging interrupt 4 (MSG 4) vector/priority register | R/W | 0x8000_0000 | 9.3.7.5/9-45 |
| 0x1690 | MIDR4—Messaging interrupt 4 (MSG 4) destination register | R/W | 0x0000_0001 | 9.3.7.6/9-46 |
| 0x16A0 | MIVPR5—Messaging interrupt 5 (MSG 5) vector/priority register | R/W | 0x8000_0000 | 9.3.7.5/9-45 |
| 0x16B0 | MIDR5—Messaging interrupt 5 (MSG 5) destination register | R/W | 0x0000_0001 | 9.3.7.6/9-46 |
| 0x16C0 | MIVPR6—Messaging interrupt 6 (MSG 6) vector/priority register | R/W | 0x8000_0000 | 9.3.7.5/9-45 |
| 0x16D0 | MIDR6—Messaging interrupt 6 (MSG 6) destination register | R/W | 0x0000_0001 | 9.3.7.6/9-46 |
| 0x16E0 | MIVPR7—Messaging interrupt 7 (MSG 7) vector/priority register | R/W | 0x8000_0000 | 9.3.7.5/9-45 |
| 0x16F0 | MIDR7—Messaging interrupt 7 (MSG 7) destination register | R/W | 0x0000_0001 | 9.3.7.6/9-46 |
| 0x1700–0x1BF0 | Reserved | — | — | — |
| 0x1C00 | MSIVPR0—Shared message signaled interrupt vector/priority register 0 | R/W | 0x8000_0000 | 9.3.6.4/9-38 |

Table A-15. PIC Interrupt Source Registers (continued)

| PIC Interrupt Source Registers—Block Base Address 0x5_0000 | | | | |
|---|---|---|---|---|
| **Offset** | **Register** | **Access** | **Reset** | **Section/Page** |
| 0x1C10 | MSIDR0—Shared message signaled interrupt destination register 0 | R/W | 0x0000_0001 | 9.3.6.5/9-39 |
| 0x1C20 | MSIVPR1—Shared message signaled interrupt vector/priority register 1 | R/W | 0x8000_0000 | 9.3.6.4/9-38 |
| 0x1C30 | MSIDR1—Shared message signaled interrupt destination register 1 | R/W | 0x0000_0001 | 9.3.6.5/9-39 |
| 0x1C40 | MSIVPR2—Shared message signaled interrupt vector/priority register 2 | R/W | 0x8000_0000 | 9.3.6.4/9-38 |
| 0x1C50 | MSIDR2—Shared message signaled interrupt destination register 2 | R/W | 0x0000_0001 | 9.3.6.5/9-39 |
| 0x1C60 | MSIVPR3—Shared message signaled interrupt vector/priority register 3 | R/W | 0x8000_0000 | 9.3.6.4/9-38 |
| 0x1C70 | MSDIR3—Shared message signaled interrupt destination register 3 | R/W | 0x0000_0001 | 9.3.6.5/9-39 |
| 0x1C80 | MSIVPR4—Shared message signaled interrupt vector/priority register 4 | R/W | 0x8000_0000 | 9.3.6.4/9-38 |
| 0x1C90 | MSIDR4—Shared message signaled interrupt destination register 4 | R/W | 0x0000_0001 | 9.3.6.5/9-39 |
| 0x1CA0 | MSIVPR5—Shared message signaled interrupt vector/priority register 5 | R/W | 0x8000_0000 | 9.3.6.4/9-38 |
| 0x1CB0 | MSIDR5—Shared message signaled interrupt destination register 5 | R/W | 0x0000_0001 | 9.3.6.5/9-39 |
| 0x1CC0 | MSIVPR6—Shared message signaled interrupt vector/priority register 6 | R/W | 0x8000_0000 | 9.3.6.4/9-38 |
| 0x1CD0 | MSIDR6—Shared message signaled interrupt destination register 6 | R/W | 0x0000_0001 | 9.3.6.5/9-39 |
| 0x1CE0 | MSIVPR7—Shared message signaled interrupt vector/priority register 7 | R/W | 0x8000_0000 | 9.3.6.4/9-38 |
| 0x1CF0 | MSDIR7—Shared message signaled interrupt destination register 7 | R/W | 0x0000_0001 | 9.3.6.5/9-39 |
| 0x1D00–0xFFF0 | Reserved | — | — | — |

### A.2.3 PIC—Processor (per-CPU) Registers

Table A-16. PIC Processor (per-CPU) Registers

| PIC Processor (per-CPU) Registers—Block Base Address 0x6_0000 | | | | |
|---|---|---|---|---|
| **Offset** | **Register** | **Access** | **Reset** | **Section/Page** |
| 0x0000–0x0030 | Reserved | — | — | — |

**Table A-16. PIC Processor (per-CPU) Registers (continued)**

| PIC Processor (per-CPU) Registers—Block Base Address 0x6_0000 | | | | |
|---|---|---|---|---|
| **Offset** | **Register** | **Access** | **Reset** | **Section/Page** |
| 0x0040 | IPIDR0—Processor core 0 interprocessor 0 dispatch register | W | all zeros | 9.3.8.1/9-48 |
| 0x0050 | IPIDR1—Processor core 0 interprocessor 1 dispatch register | | | |
| 0x0060 | IPIDR2—Processor core 0 interprocessor 2 dispatch register | | | |
| 0x0070 | IPIDR3—Processor core 0 interprocessor 3 dispatch register | | | |
| 0x0080 | CTPR0—Processor core 0 current task priority register | R/W | 0x0000_000F | 9.3.8.2/9-49 |
| 0x0090 | WHOAMI0—Processor core 0 who am I register | R | n/a | 9.3.8.3/9-50 |
| 0x00A0 | IACK0—Processor core 0 interrupt acknowledge register | R | all zeros | 9.3.8.4/9-50 |
| 0x00B0 | EOI0—Processor core 0 end of interrupt register | W | all zeros | 9.3.8.5/9-51 |
| 0x00C0–<br>0x0FF0 | Reserved | — | — | — |
| 0x1000–<br>0x1030 | Reserved | — | — | — |
| 0x1040 | IPIDR0—Processor core 1 interprocessor 0 dispatch register | W | all zeros | 9.3.8.1/9-48 |
| 0x1050 | IPIDR1—Processor core 1 interprocessor 1 dispatch register | | | |
| 0x1060 | IPIDR2—Processor core 1 interprocessor 2 dispatch register | | | |
| 0x1070 | IPIDR3—Processor core 1 interprocessor 3 dispatch register | | | |
| 0x1080 | CTPR1—Processor core 1 current task priority register | R/W | 0x0000_000F | 9.3.8.2/9-49 |
| 0x1090 | WHOAMI1—Processor core 1 who am I register | R | n/a | 9.3.8.3/9-50 |
| 0x10A0 | IACK1—Processor core 1 interrupt acknowledge register | R | all zeros | 9.3.8.4/9-50 |
| 0x10B0 | EOI1—Processor core 1 end of interrupt register | W | all zeros | 9.3.8.5/9-51 |

## A.3 Device-Specific Utilities

The device-specific utilities registers control functions that are not particular to a functional unit but to the device as a whole; they occupy the highest 256 Kbytes of CCSR space (0xE_0000–0xF_FFFF).

### A.3.1 Global Utilities

**Table A-17. Global Utilities Registers**

| Global Utilities—Block Base Address 0xE_0000 | | | | |
|---|---|---|---|---|
| **Offset** | **Register** | **Access** | **Reset** [1] | **Section/Page** |
| **Power-On Reset Configuration Values** | | | | |
| 0x000 | PORPLLSR—POR PLL Ratio Status Register | R | 0x*nnnn_nnnn* | 23.4.1.1/23-5 |
| 0x004 | PORBMSR—POR Boot Mode Status Register | R | 0x*nnnn*_0000 | 23.4.1.2/23-6 |
| 0x008 | PORIMPSCR—POR I/O Impedance Status And Control Register | Mixed | 0x000*n*_007F | 23.4.1.3/23-8 |

### Table A-17. Global Utilities Registers (continued)

| Global Utilities—Block Base Address 0xE_0000 | | | | |
|---|---|---|---|---|
| **Offset** | **Register** | **Access** | **Reset [1]** | **Section/Page** |
| 0x00C | PORDEVSR—POR Device Status Register | R | 0x*nnnn_nnn*0 | 23.4.1.4/23-9 |
| 0x010 | PORDBGMSR—POR Debug Mode Status Register | R | 0x0*n*00_0000 | 23.4.1.5/23-12 |
| 0x014 | PORDEVSR2—POR Device Status Register 2 | R | 0x*nn*00_001F | 23.4.1.6/23-13 |
| 0x020 | GPPORCR—General-purpose POR Configuration Register | R | 0x*nnnn_nnnn* | 23.4.1.7/23-13 |
| **General Configuration Controls** | | | | |
| 0x030 | GENCFGR—General Configuration Register | R/W | 0x0000_0000 | 23.4.1.8/23-14 |
| **Signal Multiplexing Controls** | | | | |
| 0x060 | PMUXCR—Alternate Function Signal Multiplex Control | R/W | 0x0000_0000 | 23.4.1.9/23-14 |
| **Device Disables** | | | | |
| 0x070 | DEVDISR—Device Disable Control | R/W | 0x*nn*0*n*_0*n*0*n* | 23.4.1.10/23-16 |
| **Power Management Registers** | | | | |
| 0x07C | PMJCR—Power Management Jog Control Register | R/W | 0x00*nn*_0000 | 23.4.1.11/23-19 |
| 0x080 | POWMGTCSR—Power Management Status And Control Register | Mixed | 0x0000_0000 | 23.4.1.12/23-21 |
| 0x084 | PMRCCR—Power Management Reset Counters Configuration Register | R/W | 0x0C83_09D1 | 23.4.1.13/23-22 |
| 0x088 | PMPDCCR—Power Management Power Down Counters Configuration Register | R/W | 0x08D1_0000 | 23.4.1.14/23-24 |
| 0x08C | PMCDR—Power Management Clock Disable Register | R/W | 0x0000_0800 | 23.4.1.15/23-25 |
| **Interrupt and Reset Status and Control** | | | | |
| 0x090 | MCPSUMR—Machine Check Summary Register | w1c | 0x0000_0000 | 23.4.1.16/23-26 |
| 0x094 | RSTRSCR—Reset Request Status And Control Register | R | 0x0000_0000 | 23.4.1.17/23-27 |
| 0x098 | ECTRSTCR—Exception Reset Control Register | Mixed | 0x0000_0000 | 23.4.1.18/23-28 |
| 0x09C | RSTSR—Automatic Reset Status Register | Mixed | 0x0000_0000 | 23.4.1.19/23-28 |
| **Version Registers** | | | | |
| 0x0A0 | PVR—Processor version register | R | e500 processor version | 23.4.1.20/23-29 |
| 0x0A4 | SVR—System version register | R | MPC8536E system version | 23.4.1.21/23-30 |
| **Status Registers** | | | | |
| 0x0B0 | RSTCR—Reset control register | R/W | 0x0000_0000 | 23.4.1.22/23-30 |

Table A-17. Global Utilities Registers (continued)

| Global Utilities—Block Base Address 0xE_0000 | | | | |
|---|---|---|---|---|
| Offset | Register | Access | Reset [1] | Section/Page |
| 0x0C0 | LBCVSELCR—LBC voltage select control register | R/W | 0x0000_0000 | 23.4.1.23/23-31 |
| 0xB28 | DDRCLKDR—DDR clock disable register | R/W | 0x0000_0000 | 23.4.1.24/23-32 |
| Debug Control Registers | | | | |
| 0xE00 | CLKOCR—Clock out control register | R/W | 0x0000_0000 | 23.4.1.25/23-33 |
| 0xE20 | ECMCR—ECM control register | R/W | 0x0000_0000 | 23.4.1.26/23-33 |
| 0xE60 | GCR—General control register | R/W | 0x0000_*n*000 | 23.4.1.27/23-34 |
| SerDes1 Registers—Block Base Address 0xE_3000 | | | | |
| 0xE_3000 | SRDS1CR0—SerDes1 control register 0 | R/W | 0x1100_4430 | 23.4.1.28/23-35 |
| 0xE_3008 | SRDS1CR2—SerDes1 control register 2 | R/W | 0x0000_0040 | 23.4.1.29/23-37 |
| SerDes2 Registers—Block Base Address 0xE_3100 | | | | |
| 0xE_3100 | SRDS2CR0—SerDes2 control register 0 | R/W | 0x1100_4430 | 23.4.1.30/23-39 |
| 0xE_3104 | SRDS2CR1—SerDes2 control register 1 | R/W | 0x0000_0040 | 23.4.1.31/23-41 |
| 0xE_3108 | SRDS2CR2—SerDes2 control register 2 | R/W | 0x0000_1C1C | 23.4.1.32/23-42 |
| 0xE_310C | SRDS2CR3—SerDes2 control register 3 | R/W | 0x0101_0000 | 23.4.1.33/23-44 |

[1] Bits indicated with *n* are set from configuration signals.

## A.3.2 Device Performance Monitor

Table A-18. Performance Monitor Registers

| Performance Monitor—Block Base Address 0xE_1000 | | | | |
|---|---|---|---|---|
| Offset | Register | Access | Reset | Section/Page |
| 0x000 | PMGC0—Performance monitor global control register | R/W | 0x0000_0000 | 24.3.2.1/24-5 |
| 0x010 | PMLCA0—Performance monitor local control register A0 | R/W | 0x0000_0000 | 24.3.2.2/24-6 |
| 0x014 | PMLCB0—Performance monitor local control register B0 | R/W | 0x0000_0000 | 24.3.2.2/24-6 |
| 0x018 | PMC0 (lower)—Performance monitor counter 0 upper | R/W | 0x0000_0000 | 24.3.3.1/24-10 |
| 0x01C | PMC0 (upper)—Performance monitor counter 0 lower | R/W | 0x0000_0000 | 24.3.3.1/24-10 |
| 0x020 | PMLCA1—Performance monitor local control register A1 | R/W | 0x0000_0000 | 24.3.2.2/24-6 |
| 0x024 | PMLCB1—Performance monitor local control register B1 | R/W | 0x0000_0000 | 24.3.2.2/24-6 |
| 0x028 | PMC1—Performance monitor counter 1 | R/W | 0x0000_0000 | 24.3.3.1/24-10 |
| 0x030 | PMLCA2—Performance monitor local control register A2 | R/W | 0x0000_0000 | 24.3.2.2/24-6 |
| 0x034 | PMLCB2—Performance monitor local control register B 2 | R/W | 0x0000_0000 | 24.3.2.2/24-6 |

Table A-18. Performance Monitor Registers (continued)

| Performance Monitor—Block Base Address 0xE_1000 | | | | |
|---|---|---|---|---|
| Offset | Register | Access | Reset | Section/Page |
| 0x038 | PMC2—Performance monitor counter 2 | R/W | 0x0000_0000 | 24.3.3.1/24-10 |
| 0x040 | PMLCA3—Performance monitor local control register A3 | R/W | 0x0000_0000 | 24.3.2.2/24-6 |
| 0x044 | PMLCB3—Performance monitor local control register B3 | R/W | 0x0000_0000 | 24.3.2.2/24-6 |
| 0x048 | PMC3—Performance monitor counter 3 | R/W | 0x0000_0000 | 24.3.3.1/24-10 |
| 0x050 | PMLCA4—Performance monitor local control register A4 | R/W | 0x0000_0000 | 24.3.2.2/24-6 |
| 0x054 | PMLCB4—Performance monitor local control register B4 | R/W | 0x0000_0000 | 24.3.2.2/24-6 |
| 0x058 | PMC4—Performance monitor counter 4 | R/W | 0x0000_0000 | 24.3.3.1/24-10 |
| 0x060 | PMLCA5—Performance monitor local control register A5 | R/W | 0x0000_0000 | 24.3.2.2/24-6 |
| 0x064 | PMLCB5—Performance monitor local control register B 5 | R/W | 0x0000_0000 | 24.3.2.2/24-6 |
| 0x068 | PMC5—Performance monitor counter 5 | R/W | 0x0000_0000 | 24.3.3.1/24-10 |
| 0x070 | PMLCA6—Performance monitor local control register A6 | R/W | 0x0000_0000 | 24.3.3.1/24-10 |
| 0x074 | PMLCB6—Performance monitor local control register B6 | R/W | 0x0000_0000 | 24.3.2.2/24-6 |
| 0x078 | PMC6—Performance monitor counter 6 | R/W | 0x0000_0000 | 24.3.3.1/24-10 |
| 0x080 | PMLCA7—Performance monitor local control register A7 | R/W | 0x0000_0000 | 24.3.2.2/24-6 |
| 0x084 | PMLCB7—Performance monitor local control register B7 | R/W | 0x0000_0000 | 24.3.2.2/24-6 |
| 0x088 | PMC7—Performance monitor counter 7 | R/W | 0x0000_0000 | 24.3.3.1/24-10 |
| 0x090 | PMLCA8—Performance monitor local control register A8 | R/W | 0x0000_0000 | 24.3.2.2/24-6 |
| 0x094 | PMLCB8—Performance monitor local control register B8 | R/W | 0x0000_0000 | 24.3.2.2/24-6 |
| 0x098 | PMC8—Performance monitor counter 8 | R/W | 0x0000_0000 | 24.3.3.1/24-10 |
| 0x0A0 | PMLCA9—Performance monitor local control register A9 | R/W | 0x0000_0000 | 24.3.2.2/24-6 |
| 0x0A4 | PMLCB9—Performance monitor local control register B9 | R/W | 0x0000_0000 | 24.3.2.2/24-6 |
| 0x0A8 | PMC9—Performance monitor counter 9 | R/W | 0x0000_0000 | 24.3.3.1/24-10 |

### A.3.3 Watchpoint Monitor and Trace Buffer

Table A-19. Watchpoint Monitor and Trace Buffer Registers

| Watchpoint Monitor and Trace Buffer—Block Base Address 0xE_2000 | | | | |
|---|---|---|---|---|
| Offset | Register | Access | Reset | Section/Page |
| Watchpoint Monitor Registers | | | | |
| 0x000 | WMCR0—Watchpoint monitor control register 0 | R/W | 0x0000_0000 | 25.3.1.1/25-10 |
| 0x004 | WMCR1—Watchpoint monitor control register 1 | R/W | 0x0000_0000 | 25.3.1.1/25-10 |
| 0x00C | WMAR—Watchpoint monitor address register | R/W | 0x0000_0000 | 25.3.1.2/25-12 |

Table A-19. Watchpoint Monitor and Trace Buffer Registers (continued)

| Watchpoint Monitor and Trace Buffer—Block Base Address 0xE_2000 | | | | |
|---|---|---|---|---|
| **Offset** | **Register** | **Access** | **Reset** | **Section/Page** |
| 0x014 | WMAMR—Watchpoint monitor address mask register | R/W | 0x0000_0000 | 25.3.1.3/25-13 |
| 0x018 | WMTMR—Watchpoint monitor transaction mask register | R/W | 0x0000_0000 | 25.3.1.4/25-13 |
| 0x01C | WMSR—Watchpoint monitor status register | R/W | 0x0000_0000 | 25.3.1.5/25-15 |
| **Trace Buffer Registers** | | | | |
| 0x040 | TBCR0—Trace buffer control register 0 | R/W | 0x0000_0000 | 25.3.2.1/25-15 |
| 0x044 | TBCR1—Trace buffer control register 1 | R/W | 0x0000_0000 | 25.3.2.1/25-15 |
| 0x04C | TBAR—Trace buffer address register | R/W | 0x0000_0000 | 25.3.2.2/25-18 |
| 0x054 | TBAMR—Trace buffer address mask register | R/W | 0x0000_0000 | 25.3.2.3/25-18 |
| 0x058 | TBTMR—Trace buffer transaction mask register | R/W | 0x0000_0000 | 25.3.2.4/25-19 |
| 0x05C | TBSR—Trace buffer status register | R/W | 0x0000_0000 | 25.3.2.5/25-19 |
| 0x060 | TBACR—Trace buffer access control register | R/W | 0x0000_0000 | 25.3.2.6/25-20 |
| 0x064 | TBADHR—Trace buffer access data high register | R/W | 0x0000_0000 | 25.3.2.7/25-21 |
| 0x068 | TBADR—Trace buffer access data register | R/W | 0x0000_0000 | 25.3.2.8/25-21 |
| **Context ID Registers** | | | | |
| 0x0A0 | PCIDR—Programmed context ID register | R/W | 0x0000_0000 | 25.3.3.1/25-22 |
| 0x0A4 | CCIDR—Current context ID register | R/W | 0x0000_0000 | 25.3.3.2/25-23 |
| **Other Registers** | | | | |
| 0x0B0 | TOSR—Trigger output source register | R/W | 0x0000_0000 | 25.3.4.1/25-23 |

# Appendix B
# Revision History

This appendix provides a list of major differences between revisions of the *MPC8536E PowerQUICC III Integrated Processor Reference Manual*.

## B.1 Changes From Revision 0 to Revision 1

Major changes from Revision 0 to Revision 1 are as follows:

| Section/Page | Changes |
|---|---|
| 4.3.1.1.2, 4-5 | Added clarification for CCSRBAR[BASE_ADDR] as follows:<br>Replaced phrase “identifies the16 most-significant address bits of the window” with “identifies the16 most-significant address bits of the 36-bit window.” |
| 4.4.4.1, 4-25 | Added the following sentence to second paragraph:<br>“If the separate (asynchronous) PCI_CLK clock signal is used rather than SYSCLK as the PCI clock, then this clock must be constantly driven, even when in Deep Sleep mode in order to avoid loss of lock.” |
| 4.4.4.2, 4-25 | Added the following sentence to end of section:<br>“For any SerDes that is not disabled through cfg_io_ports[0:2]=001 or cfg_srds2_prtcl[0:2]=111 respectively, the applicable SDn_REF_CLK/SDn_REF_CLK must be constantly driven, even when in Deep Sleep mode, in order to avoid loss of lock.” |
| 5.3.1, 5-6 | Updated SVR values in Table 5-2, “Device Revision Level Cross-Reference,” as follows:<br>0x803F_0091 for MPC8536E Rev 1.1 (with security)<br>0x8037_0091 for MPC8536 Rev 1.1 (without security). |
| 8.3.2, 8-7 | Changed signal description of MA[15:0] from:<br>Assertion/Negation—The address is always driven when the memory controller is enabled. It is valid when a transaction is driven to DRAM (when $\overline{\text{MCS}}$ is active).<br>to:<br>Assertion/Negation—The address lines are only driven when the controller has a command scheduled to issue on the address/CMD bus; otherwise they will be at high-Z. It is valid when a transaction is driven to DRAM (when $\overline{\text{MCS}}$ is active). |
| 8.4.1.9, 8-26 | Updated DDR_SDRAM_CFG_2[DQS_CFG] field description to designate a value of 0x0 as reserved. (Note that since the the default value for this field is reserved, software must configure this field to a valid value during initialization.) |

8.4.1.27, 8-47 Added note to this section:

All driver calibration, whether by software or hardware, should be done before the DDR controller is enabled (before DDR_SDRAM_CFG[MEM_EN] is set).

8.4.1.27, 8-47 In Table 8-33, “DDRCDR_1 Field Descriptions,” updated some settings for field ODT as follows:

001: changed from 46 Ohms to 55 Ohms
011: changed from 43 Ohms to 50 Ohms
101: changed from 33 Ohms to 43 Ohms

Also added clarification to DDRCDR_1[ODT] that the ODT value (which is obtained by concatenating DDRCDR_1[ODT] and DDRCDR_2[ODT]) is obtained as follows:

Note that the order of concatenation is (from left to right)
DDRCDR_1[ODT], DDRCDR_2[ODT]

8.4.1.28, 8-50 In Table 8-34, “DDRCDR_2 Field Descriptions,” updated some settings for field ODT as follows:

001: changed from 46 Ohms to 55 Ohms
011: changed from 43 Ohms to 50 Ohms
101: changed from 33 Ohms to 43 Ohms

Also added clarification to DDRCDR_2[ODT] that the ODT value (which is obtained by concatenating DDRCDR_1[ODT] and DDRCDR_2[ODT]) is obtained as follows:

Note that the order of concatenation is (from left to right)
DDRCDR_1[ODT], DDRCDR_2[ODT]

8.6.1, 8-91 Updated DQS_CFG configuration row. Specifically, DDR2 configuration formerly read:

‘Can be set to either 00 or 01, depending on if differential strobes are used’

now reads:
‘Should be set to 01’

8.6.2, 8-93 Added clarification that DDR3 specification requires additional delay by adding the phrase “500 ms for DDR3” to the second sentence of the first paragraph of this section, as follows:

“Note that 200 ms (500 ms for DDR3) must elapse after DRAM clocks are stable...”

9.3.1.3, 9-20 Changed reset value of FRR[NIRQ] from 0x6B to 0x67

9.3.7.6, 9-46 Removed fields EP, CI0 and CI1

10.3.5, 10-30 In Table 10-10, “Descriptor Format Summary,” updated row for KEU f9

10.7.1.11., 10-74 Added note that AES-CCM does not support zero-length AAD and payload simultaneously

13.4.1.8, 13-48 Updated Figure 13-33, “eLBC Bus Cycles in PLL and PLL-bypassed Modes (GPCM and UPM only),” to show LCSn deasserted one-half LCLK cycle later

| | |
|---|---|
| 13.3.1.15, 13-33 | Corrected the LBCR[AHD] field state description as follows:<br>0 During address phases on the local bus, the LALE signal negates one platform clock period prior to the address being invalidated. For instance, at 33.3 MHz, this provides 3 ns of additional address hold time at the external address latch.<br>1 During address phases on the local bus, the LALE signal negates 0.5 platform clock period prior to the address being invalidated. This halves the address hold time, but extends the latch enable duration. This may be necessary for very high frequency designs. |
| 13.4.2, 13-49 | Corrected Figure 13-33 so that LAD[0:31] of the eLBC comes out and [12:26] goes to the Latch and then connects to A[19:5] in the Memory/Peripheral. |
| 13.5.1.1 | Removed section |
| 13.5.4.4, 13-98 | Corrected phrase 'The sequence is initiated by writing FMR[OP] = 10' to read 'The sequence is initiated by writing FMR[OP]=11' |
| 13.5.4.5, 13-98 | Corrected phrase 'The sequence is initiated by writing FMR[OP] = 10' to read 'The sequence is initiated by writing FMR[OP]=11' |
| 13.5.4.6, 13-99 | Corrected phrase 'The sequence is initiated by writing FMR[OP] = 10' to read 'The sequence is initiated by writing FMR[OP]=11' |
| 14.2, 14-4 | Added clarification to 1588 features bullet item as follows:<br>(1588 not supported in conjunction with SGMII 10/100) |
| 14.5.3.1.3, 14-30 | Changed second sentence of IEVENT[CRL] field description from:<br>The frame is discarded without being transmitted and transmission of the next frame commences.<br>to:<br>The frame is discarded without being transmitted and the queue halts (TSTAT[THLT*n*] set to 1)." |
| 14.5.3.1.6, 14-35 | Updated ECNTRL[CLRCNT] field description to read as follows:<br>Clear all statistics counters and carry registers.<br>0 Allow MIB counters to continue to increment and keep any overflow indicators.<br>1 Reset all MIB counters and CAR1 and CAR2.<br>This bit is self-resetting.<br>Updated ECNTRL[AUTOZ] field description to read as follows:<br>Automatically zero MIB counter values and carry registers.<br>0 The user must write the addressed counter zero after a host read.<br>1 The addressed counter value is automatically cleared to zero after a host read.<br>This is a steady state signal and must be set prior to enabling the Ethernet controller and must not be changed without proper care. |

14.5.3.1.8, 14-39 Corrected DMACTRL[TOD] field definition by replacing the “1” definition with the following:

1 eTSEC immediately fetches a new TxBD from ring 0.

14.5.3.2.1, 14-42 Changed TCTRL[TXSCHED] field description for 01 state to read as follows:

01 Priority scheduling mode. Frames from enabled TxBD rings are serviced in ascending ring index order.

14.5.3.5.1, 14-74 Added the following note to descriptions of MACCFG1 fields Tx_Flow and Rx_Flow:

**Note:** Should not be set when operating in Half-Duplex mode

14.5.3.5.2, 14-76 In MACCFG2[Huge Frame] field description, updated the right-hand “Buffer descriptor updated” column as follows:

| Frame type | ... | Buffer descriptor updated |
|---|---|---|
| Receive or transmit | ... | yes |
| Receive | ... | no |
| Transmit | ... | yes |
| Receive or transmit | ... | no |

14.5.3.5.5, 14-79 Replaced first paragraph of field description of Maximum Frame with the following:

This field is set to 0x0600 (1536 bytes) by default and always must be set to a value greater than or equal to 0x0040 (64 bytes), but not greater than 0x2580 (9600 bytes). It sets the maximum Ethernet frame size in both the transmit and receive directions. (Refer to MACCFG2[Huge Frame].) It does not affect the size of packets sent or received via the FIFO packet interface.

14.5.3.6, 14-87 Added note to end of section:

The transmit and receive frame counters (TR64, TR127, TR 255, TR511, TR1K, TRMAX, adn TRMGV) do not increment for aborted frames (collision retry limit exceeded, late collision, underrun, EBERR, TxFIFO data error, frame truncated due to exceeding MAXFRM, or excessive deferral).

14.5.3.6.25, 14-99 Replaced second sentence of TBYT[TBYT] with the following:

This count does not include preamble/SFD or jam bytes, except for half-duplex flow control (back-pressure triggered by TCTRL[THDF]=1). For THDF, the sum total of ‘phantom’ preamble bytes transmitted for flow control purposes is included in the TBYT increment value of the next frame to be transmitted, up to 65,535 bytes of frame and phantom preamble.

14.5.3.6.41, 14-107 Replaced description of TOVR[TOVR] with the following:

Transmit oversize frame counter. Increments each time a frame is transmitted which exceeds 1518 (non VLAN) or 1522 (VLAN) with a correct FCS value.

14.5.3.6.44, 14-109

14.5.3.6.45, 14-110 Corrected access designation for CAR1 and CAR2 registers to be ‘w1c’

14.5.3.9.2, 14-119 Replaced ATTRELI[EI] field description with the following:

Extracted index. Points to the first byte, as a multiple of 64 bytes, within the receive frame as sent to memory from which to begin extracting data.

14.5.3.10.2, 14-121 Corrected RFBPTR0–RFBPTR7 register offset designation to read as follows:

eTSEC1:0x2_4C44+8×*n*; eTSEC2:0x2_5C44+8×*n*;
eTSEC3:0x2_6C44+8×*n*; eTSEC4:0x2_7C44+8×*n*”

14.5.3.11.1, 14-123 Replaced TMR_CTRL[CIPH] field description with the following:

Oscillator input clock phase.
0 non-inverted timer input clock
1 inverted timer input clock (NOTE: this setting is reserved if CKSEL=01.)

14.5.3.11.9, 14-129 Changed access of register TMR_ACC from read only to read/write.

14.5.3.11.12, 14-131 Changed access of register TMR_ALARM1–2_H/L from mixed to read/write

14.5.3.11.13, 14-132 Changed access of register TMR_FIPER1–3 from mixed to read/write

14.6.3.9, 14-171 Replaced second sentence of third paragraph (began “Since the pause timer commences counting...”) with the following:

The controller completes any frame in progress before stopping transmission and does not commence counting the pause time until transmit is idle.

14.6.5.3.1, 14-190 Replaced entire section, “Priority-Based Queuing (PBQ),” with the following:

PBQ is the simplest scheduler decision policy. The enabled TxBD rings are assigned a priority value based on their index. Rings with a lower index have precedence over rings with higher indices, with priority assessed on a frame-by-frame basis. For example, frames in TxBD ring 0 have higher priority than frames in TxBD ring 1, and frames in TxBD ring 1 have higher priority than frames in TxBD ring 2, and so on.

The scheduling decision is then achieved as follows:

```
loop
     # start or S/W clear of TSATn
     ring = 0;
     while ring <= 7 loop
      if enabled(ring) and not ring_empty(ring) then
             transmit_frame(ring);
             ring = 0;
      else
             ring = ring + 1;
      endif
     endloop
endloop
```

14.6.7, 14-194 Added the following note after third paragraph of this section:

IEEE 1588 timestamping is not supported in conjunction with the SGMII 10/100 interface mode.“

14.7.1.8, 14-234 Added the following note:

SGMII mode utilizes the internal TBI PHY. The internal TBI PHY only auto-negotiates at 1 Gbps. However, 10 Mbps and 100 Mbps speeds are supported in SGMII mode. It is recommended that the external PHY inform the MAC if the desired link speed is not 1 Gbps. Software can perform MII management cycles to determine the external PHY link speed and program ECNTRL and MACCFG2 accordingly.

17.3.10.5, 17-86 Changed access of PCI Express Correctable Error Status Register from read/write to w1c.

17.4.1.8, 17-103 Revised first paragraph by deleting “... originating from the PCI Express outbound ATMUs,” from the first sentence. Also, deleted the last sentence that stated, "Note that configuration writes originating from the PCI Express configuration access registers (PEX_CONFIG_ADDR/PEX_CONFIG_DATA) are not serialized."

18.1, 18-1 Changed Figure 19-1, “eSPI Block Diagram,” to show input clock labeled as “CCB clock divided by 2” rather than “system clock.”

18.3.1.7, 18-12 Added note to SPMODE*n* fields DIV16*n* and PM*n* as follows:

“System clock as used here is defined to be CCB clock divided by 2.”

19.3.3.2, 19-16 In register SError (SATA Interface Error Register), made bits 10 and 24 Reserved.

20.4.17, 20-36 For HOSTVER[VVN], added “0x01 Freescale eSDHC version 2.0”

20.6.5, 20-58 In Table 20-27, “Commands for MMC/SD,” modified Argument column entry for ACMD23 to read as follows:

[31:23] stuff bits
[22:0] number of blocks”

20.6.6, 50-59 Modified existing note in section to read as follows:

“When the internal DMA is not enabled and a write transaction is in operation, DATPORT must not be read. DATPORT also must not be used to read (or write) data by the CPU or external DMA if the data will be written (or read) by the eSDHC internal DMA.”

21.3.2.11, 21-22 In description of TXFILLTUNING[TXSCHOH], changed formula (introduction reads “A good value to begin with is:”) as follows:

TXFIFOTHRES × (BURSTSIZE × 4 bytes-per-word) ÷ (40 × TimeUnit)

(Formerly contained incorrect term BURSTSIZE ÷ 4 bytes-per-word)

23.4.1.8, 23-14 Added new register, GENCFGR, “General Configuration Register”

23.4.1.20, 23-29 Updated SVR values for silicon revision 1.1 as follows:

0x803F_0091 for MPC8536E Rev 1.1 (with security)
0x8037_0091 for MPC8536 Rev 1.1 (without security).

23.4.1.28, 23-37 Feplaced recommended setting for PCI Express in SRDS1CR2[X3SA–X3SF] (8 bits in all) with “0” (that is, not disabled)

# Appendix C
# MPC8535E

This appendix provides a list of major differences between the MPC8536E and the MPC8535E.

## C.1 Overview of Differences

Table C-1 summarizes the differences between the MPC8536E and the MPC8535E. The remainder of this appendix further clarifies these differences.

**Table C-1. Comparison of Features, MPC8536E and MPC8535E**

| Feature | | MPC8536E | MPC8535E |
|---|---|---|---|
| PCI Express | Interfaces | 3 | 2 |
| | Maximum width | x8 | x4 |
| USB | | 3 | 2 |
| SATA | | 2 | 1 |
| SGMII | Interfaces | 2 | 1 |
| | Location | eTSEC1, eTSEC3 | eTSEC1 |

Figure C-1 shows the major functional units within the MPC8535E.

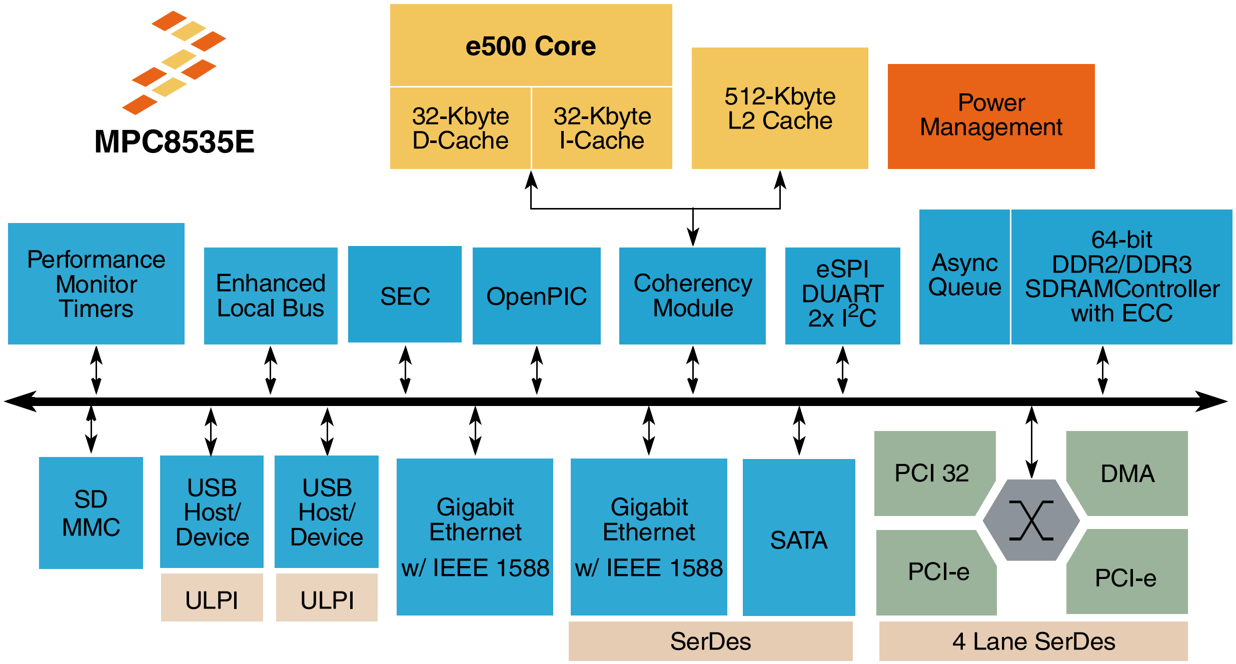


**Figure C-1. MPC8535E Block Diagram**

## C.2 Signal Differences

The following signal functionality described in the MPC8536E Reference Manual is not available in the MPC8535E during normal operation:

- PCI Express interface—SD1_TX[0:3] and $\overline{\text{SD1\_TX}}$[0:3]
- PCI Express interface—SD1_RX[0:3] and $\overline{\text{SD1\_RX}}$[0:3]
- SATA and SGMII—SD2_RX1, $\overline{\text{SD2\_RX1}}$, SD2_TX1, $\overline{\text{SD2\_TX1}}$
- USB3—USB3_D[7:0], USB3_NXT, USB3_DIR, USB3_STP, and USB3_CLK

Please refer to the *MPC8536E Integrated Processor Hardware Specifications* for details on remaining functionality of associated signals, including power-on reset configuration functionality.

## C.3 Reset/Configuration Differences

This section details significant differences to reset/configuation functionality on the MPC8535E. The reader should consult the *MPC8536E Integrated Processor Hardware Specifications* for further details.

### C.3.1 Boot ROM Location

MPC8535E boot ROM location options differ slightly from those of the MPC8536E (as described in Section 4.4.3.6, "Boot ROM Location").The boot ROM location inputs, shown in Table C-2, select the physical location of boot ROM for the MPC8535E.

**Table C-2. Boot ROM Location (MPC8535E)**

| Functional Signals | Reset Configuration Name | Value (Binary) | Meaning |
|---|---|---|---|
| TSEC1_TXD[7:4]<br>Default (1111) | cfg_rom_loc[0:3] | 0000 | PCI |
| | | 0001 | Reserved |
| | | 0010 | PCI Express 2 |
| | | 0011 | PCI Express 3 |
| | | 0100 | DDR controller |
| | | 0101 | Reserved |
| | | 0110 | On-chip boot ROM eSPI configuration |
| | | 0111 | On-chip boot ROM eSDHC configuration |
| | | 1000 | Local bus FCM—8-bit NAND Flash small page ECC enabled |
| | | 1001 | Local bus FCM—8-bit NAND Flash small page ECC disabled |
| | | 1010 | Local bus FCM—8-bit NAND Flash large page ECC enabled |
| | | 1011 | Local bus FCM—8-bit NAND Flash large page ECC disabled |
| | | 1100 | Reserved |
| | | 1101 | Local bus GPCM—8-bit ROM |
| | | 1110 | Local bus GPCM—16-bit ROM |
| | | 1111 | Local bus GPCM—32-bit ROM (default) |

## C.3.2 Host/Agent Configuration

MPC8535E host/agent configuration options differ slightly from those of the MPC8536E (as described in Section 4.4.3.7, “Host/Agent Configuration”). The MPC8535E options are shown in Table C-3.

**Table C-3. Host/Agent Configuration (MPC8535E)**

| Functional Signals | Reset Configuration Name | Value (Binary) | Meaning |
|---|---|---|---|
| $\overline{\text{LWE}}$[1:3]/$\overline{\text{LBS}}$[1:3]<br>Default (111) | cfg_host_agt[0:2] | 000 | Reserved |
| | | 001 | MPC8536E acts as an endpoint on PCI Express 3 interface. It acts as the host/root complex for the PCI and PCI Express 2 interfaces. |
| | | 010 | Reserved |
| | | 011 | MPC8536E acts as an endpoint on PCI Express 2 interface. It acts as the host/root complex for the PCI and PCI Express 3 interfaces. |
| | | 100 | Reserved |
| | | 101 | Reserved |
| | | 110 | MPC8536E acts as an agent of an external host on its PCI interface. It acts as a root complex for both PCI Express interfaces. |
| | | 111 | MPC8536E acts as the host processor/root complex on all interfaces (default). |

## C.3.3 I/O Port Selection

Because the MPC8535E has fewer PCI Express and SATA interfaces and does not support SGMII, I/O port selection, as described in Section 4.4.3.8, “SerDes1 I/O Port Selection and Section 4.4.3.9, “SerDes2 I/O Port Selection,” differs between the two devices. This section describes these differences.

### C.3.3.1 SerDes1 (PCI Express) I/O Port Selection

Table C-4 shows the configuration of I/O ports and bit rates (and required reference clocks) that are possible for the SerDes1 interfaces.

**Table C-4. SerDes1 I/O Port Selection (MPC8535E)**

| Functional Signal | Reset Configuration Name | Value (Binary) | Meaning |
|---|---|---|---|
| TSEC3_TXD[6:4]<br>Default (111) | cfg_io_ports[0:2] | 000 | Reserved |
| | | 001 | Both PCI Express ports powered down |
| | | 010 | Reserved |
| | | 011 | Reserved |
| | | 100 | Reserved |
| | | 101 | PCI Express 2 (x4) (2.5 Gbps) → SerDes1 Lanes E–H |
| | | 110 | Reserved |
| | | 111 | PCI Express 2 (x2) (2.5 Gbps) → SerDes1 Lanes E–F<br>PCI Express 3 (x2) (2.5 Gbps) → SerDes1 Lanes G–H |

### C.3.3.2 SerDes2 (SATA) I/O Port Selection

Table C-5 shows the configuration of I/O ports and bit rates (and required reference clocks) that are possible for the SerDes2 interfaces.

**Table C-5. SerDes2 I/O Port Selection (MPC8535E)**

| Functional Signal | Reset Configuration Name | Value (Binary) | Meaning |
|---|---|---|---|
| TSEC1_TXD2, TSEC3_TXD2, TSEC_1588_ PULSE_OUT1<br>Default (111) | cfg_srds2_prtcl[0:2] | 000 | Reserved |
| | | 001 | Reserved |
| | | 010 | Reserved |
| | | 011 | SATA1 → SerDes2 Lane A.<br>eTSEC1 Ethernet interface uses parallel interface according to POR config input cfg_tsec1_prtcl. |
| | | 100 | Reserved |
| | | 101 | Reserved |
| | | 110 | SATA1 disabled.<br>eTSEC1 SGMII (1.25 Gbps) -> Serdes2 Lane A (POR config input cfg_tsec1_prtcl should be left in its default setting). |
| | | 111 | SATA1 disabled.<br>eTSEC1 Ethernet interface uses parallel interface according to POR config input cfg_tsec1_prtcl.<br>Serdes2 disabled (default) |

## C.4 Differences in Peripheral Blocks

Unless specifically mentioned in the following sections, all peripheral blocks in the MPC8535E are identical to those of the MPC8536E.

### C.4.1 PCI Express Interfaces

The MPC8535E supports two PCI Express interfaces, PCI Express 2 and PCI Express 3, with maximum width of 4 bits. One of the configurations in Table C-6 can be selected during power-on reset as described in Section C.3.3.1, “SerDes1 (PCI Express) I/O Port Selection.”

**Table C-6. Supported SerDes 1 (PCI Express) Configurations**

<table>
<tr><th colspan="4">PCI Express Signal/Lane</th></tr>
<tr><th>4/E</th><th>5/F</th><th>6/G</th><th>7/H</th></tr>
<tr><td colspan="4">PEX2 x4</td></tr>
<tr><td colspan="2">PEX2 x2</td><td colspan="2">PEX3 x2</td></tr>
</table>

### C.4.2 USB Controllers

The MPC8535E supports two fully functional USB controllers, USB1 and USB2, that are compatible with USB specification revision 2.0.

### C.4.3 SATA Controllers

The MPC8535E supports one SATA controller, SATA1, that is identical to those of the MPC8536E.

### C.4.4 eTSEC Controllers

The MPC8535E supports SGMII on eTSEC1 only; therefore, the sections in the eTSEC chapter relating to SGMII on eTSEC3 are not applicable to the MPC8535E. Otherwise, the eTSEC controllers of the MPC8535E are identical to those of the MPC8536E.

# Glossary

The glossary contains an alphabetical list of terms, phrases, and abbreviations used in this reference manual.

## A

**Architecture.** A detailed specification of requirements for a processor or computer system. It does not specify details of how the processor or computer system must be implemented; instead it provides a template for a family of compatible *implementations*.

**Atomic access.** A bus access that attempts to be part of a read-write operation to the same address uninterrupted by any other access to that address (the term refers to the fact that the transactions are indivisible). The Power Architecture technology implements atomic accesses through the **lwarx**/**stwcx.** instruction pair.

**Autobaud.** The process of determining a serial data rate by timing the width of a single bit.

## B

**Beat.** A single state on the bus interface that may extend across multiple bus cycles. A transaction can be composed of multiple address or data *beats*.

**Big-endian.** A byte-ordering method in memory where the address *n* of a word corresponds to the *most-significant byte*. In an addressed memory word, the bytes are ordered (left to right) 0, 1, 2, 3, with 0 being the *most-significant byte*. See *Little-endian*.

**Boundedly undefined.** A characteristic of certain operation results that are not rigidly prescribed by the Power Architecture technology. Boundedly-undefined results for a given operation may vary among implementations and between execution attempts in the same implementation.

Although the architecture does not prescribe the exact behavior for when results are allowed to be boundedly undefined, the results of executing instructions in contexts where results are allowed to be boundedly undefined are constrained to ones that could have been achieved by executing an arbitrary sequence of defined instructions, in valid form, starting in the state the machine was in before attempting to execute the given instruction.

**Breakpoint.** A programmable event that forces the core to take a breakpoint exception.

**Burst.** A multiple-beat data transfer whose total size is typically equal to a cache block.

**Bus clock.** Clock that causes the bus state transitions.

**Bus master.** The owner of the address or data bus; the device that initiates or requests the transaction.

C

**Cache.** High-speed memory containing recently accessed data or instructions (subset of main memory).

**Cache block.** A small region of contiguous memory that is copied from memory into a *cache*. The size of a cache block may vary among processors; the maximum block size is one *page*. In Power Architecture processors, *cache coherency* is maintained on a cache-block basis. Note that the term 'cache block' is often used interchangeably with 'cache line.'

**Cache coherency.** An attribute wherein an accurate and common view of memory is provided to all devices that share the same memory system. Caches are coherent if a processor performing a read from its cache is supplied with data corresponding to the most recent value written to memory or to another processor's cache.

**Cache flush.** An operation that removes from a cache any data from a specified address range. This operation ensures that any modified data within the specified address range is written back to main memory. This operation is generated typically by a Data Cache Block Flush (**dcbf**) instruction.

**Caching-inhibited.** A memory update policy in which the *cache* is bypassed and the load or store is performed to or from main memory.

**Cast out.** A *cache block* that must be written to memory when a cache miss causes a cache block to be replaced.

**Changed bit.** One of two *page history bits* found in each *page table entry* (PTE). The processor sets the changed bit if any store is performed into the *page*. See also *Page access history bits* and *Referenced bit*.

**Clean.** An operation that causes a cache block to be written to memory, if modified, and then left in a valid, unmodified state in the cache.

**Clear.** To cause a bit or bit field to register a value of zero. See also *Set*.

**Context synchronization.** An operation that ensures that all instructions in execution complete past the point where they can produce an *exception*, that all instructions in execution complete in the context in which they began execution, and that all subsequent instructions are *fetched* and executed in the new context. Context synchronization may result from executing specific instructions (such as **isync** or **rfi**) or when certain events occur (such as an exception).

**Copy-back operation.** A cache operation in which a cache line is copied back to memory to enforce cache coherency. Copy-back operations consist of snoop push-out operations and cache cast-out operations.

## D

**Direct-mapped cache.** A cache in which each main memory address can appear in only one location within the cache; operates more quickly when the memory request is a cache hit.

**Double data rate.** Memory that allows data transfers at the start and end of a clock cycle. thereby doubling the data rate.

## E

**Effective address (EA).** The 32-bit address specified for a load, store, or an instruction fetch. This address is then submitted to the MMU for translation to either a *physical memory* address or an I/O address.

**Exclusive state.** MEI state (E) in which only one caching device contains data that is also in system memory.

## F

**Fetch.** Retrieving instructions from either the cache or main memory and placing them into the instruction queue.

**Flush.** An operation that causes a cache block to be invalidated and the data, if modified, to be written to memory.

**Frame-check sequence (FCS).** Specifies the standard 32-bit cyclic redundancy check (CRC) obtained using the standard CCITT-CRC polynomial on all fields except the preamble, SFD, and CRC.

## G

**General-purpose register (GPR).** Any of the 32 registers in the general-purpose register file. These registers provide the source operands and destination results for all integer data manipulation instructions. Integer load instructions move data from memory to GPRs and store instructions move data from GPRs to memory.

**Guarded.** The guarded attribute pertains to out-of-order execution. When a page is designated as guarded, instructions and data cannot be accessed out-of-order.

## H

**Harvard architecture.** An architectural model featuring separate caches and other memory management resources for instructions and data.

## I

**Illegal instructions.** A class of instructions that are not implemented for a particular processor. These include instructions not defined by the architecture. In addition, for 32-bit implementations, instructions that are defined only for 64-bit implementations are considered to be illegal instructions. For 64-bit implementations instructions that are defined only for 32-bit implementations are considered to be illegal instructions.

**Implementation.** A particular processor that conforms to the architecture, but may differ from other architecture-compliant implementations for example in design, feature set, and implementation of *optional* features.

**Inbound ATMU windows.** Mappings that perform address translation from the external address space to the local address space, attach attributes and transaction types to the transaction, and map the transaction to its target interface.

**In-order.** An aspect of an operation that adheres to a sequential model. An operation is said to be performed in-order if, at the time that it is performed, it is known to be required by the sequential execution model.

**Integer unit.** An execution unit in the core responsible for executing integer instructions.

**Inter-packet gap.** The gap between the end of one Ethernet packet and the beginning of the next transmitted packet.

**Instruction latency.** The total number of clock cycles necessary to execute an instruction and make ready the results of that instruction.

## K

**Kill.** An operation that causes a *cache block* to be invalidated without writing any modified data to memory.

## L

**L2 cache.** Level-2 cache. See *Secondary cache*.

**Latency.** The number of clock cycles necessary to execute an instruction and make ready the results of that execution for a subsequent instruction.

**Least-significant bit (lsb).** The bit of least value in an address, register, field, data element, or instruction encoding.

**Least-significant byte (LSB).** The byte of least value in an address, register, data element, or instruction encoding.

**Little-endian.** A byte-ordering method in memory where the address *n* of a word corresponds to the *least-significant byte*. In an addressed memory word, the bytes are ordered (left to right) 3, 2, 1, 0, with 3 being the *most-significant byte*. See *Big-endian*.

**Local access window.** Mapping used to translate a region of memory to a particular target interface, such as the DDR SDRAM controller or the PCI controller. The local memory map is defined by a set of eight local access windows. The size of each window can be configured from 4 Kbytes to 2 Gbytes.

## M

**Media access control (MAC) sublayer.** Sublayer that provides a logical connection between the MAC and its peer station. Its primary responsibility is to initialize, control, and manage the connection with the peer station.

**Media-independent interface (MII) sublayer.** Sublayer that provides a standard interface between the MAC layer and the physical layer for 10/100-Mbps operations. It isolates the MAC layer and the physical layer, enabling the MAC layer to be used with various implementations of the physical layer.

**Medium-dependent interface (MDI) sublayer.** Sublayer that defines different connector types for different physical media and PMD devices.

**Memory access ordering.** The specific order in which the processor performs load and store memory accesses and the order in which those accesses complete.

**Memory-mapped accesses.** Accesses whose addresses use the page or block address translation mechanisms provided by the MMU and that occur externally with the bus protocol defined for memory.

**Memory coherency.** An aspect of caching in which it is ensured that an accurate view of memory is provided to all devices that share system memory.

**Memory consistency.** Refers to agreement of levels of memory with respect to a single processor and system memory (for example, on-chip cache, secondary cache, and system memory).

**Memory management unit (MMU).** The functional unit that is capable of translating an *effective* (logical) *address* to a physical address, providing protection mechanisms, and defining caching methods.

**Modified/exclusive/invalid (MEI).** *Cache coherency* protocol used to manage caches on different devices that share a memory system. Note that neither the PowerPC ISA nor the Power ISA definitions specifies the implementation of an MEI protocol to ensure cache coherency.

**Modified state.** MEI state (M) in which one, and only one, caching device has the valid data for that address. The data at this address in external memory is not valid.

**Most-significant bit (msb).** The highest-order bit in an address, registers, data element, or instruction encoding.

**Most-significant byte (MSB).** The highest-order byte in an address, registers, data element, or instruction encoding.

## N

**NaN.** An abbreviation for not a number; a symbolic entity encoded in floating-point format. There are two types of NaNs—signaling NaNs and quiet NaNs.

**No-op.** No-operation. A single-cycle operation that does not affect registers or generate bus activity.

## O

**OCeaN.** (On-chip network) Non-blocking crossbar switch fabric. Enables full duplex port connections at 128Gb/s concurrent throughput and independent per port transaction queuing and flow control. Permits high bandwidth, high performance, as well as the execution of multiple data transactions.

**Outbound ATMU windows.** Mappings that perform address translations from local 32-bit address space to the address spaces of, which may be much larger than the local space. Outbound ATMU windows also map attributes such as transaction type or priority level.

## P

**Packet.** A unit of binary data that can be routed through a network. Sometimes packet is used to refer to the frame plus the preamble and start frame delimiter (SFD).

**Page.** A region in memory. The OEA defines a page as a 4-Kbyte area of memory aligned on a 4-Kbyte boundary.

**Page access history bits.** The *changed* and *referenced* bits in the PTE keep track of the access history within the page. The referenced bit is set by the MMU whenever the page is accessed for a read or write operation. The changed bit is set when the page is stored into. See *Changed bit* and *Referenced bit*.

**Page fault.** A page fault is a condition that occurs when the processor attempts to access a memory location that does not reside within a *page* not currently resident in *physical memory*. A page fault exception condition occurs when a matching, valid *page table entry* (PTE[V] = 1) cannot be located.

**Page table.** A table in memory is comprised of *page table entries*, or PTEs. It is further organized into eight PTEs per PTEG (page table entry group). The number of PTEGs in the page table depends on the size of the page table (as specified in the SDR1 register).

**Page table entry (PTE).** Data structures containing information used to translate *effective address* to physical address on a 4-Kbyte page basis. A PTE consists of 8 bytes of information in a 32-bit processor and 16 bytes of information in a 64-bit processor.

**Physical coding sublayer (PCS).** Sublayer responsible for encoding and decoding data stream to and from the MAC sublayer.

**Physical medium attachment (PMA) sublayer.** Sublayer responsible for serializing code groups into a bit stream suitable for serial bit-oriented physical devices (SERDES) and vice versa. Synchronization is also performed for proper data decoding in this sublayer. The PMA sits between the PCS and the PMD sublayers.

**Physical medium dependent (PMD) sublayer.** Sublayer responsible for signal transmission. The typical PMD functionality includes amplifier, modulation, and wave shaping. Different PMD devices may support different media.

**Physical memory.** The actual memory that can be accessed through the system's memory bus.

**Pipelining.** A technique that breaks operations, such as instruction processing or bus transactions, into smaller distinct stages or tenures (respectively) so that a subsequent operation can begin before the previous one has completed.

**Primary opcode.** The most-significant 6 bits (bits 0–5) of the instruction encoding that identifies the type of instruction.

**Program order.** The order of instructions in an executing program. More specifically, this term is used to refer to the original order in which program instructions are fetched into the instruction queue from the cache.

**Protection boundary.** A boundary between *protection domains*.

**Protection domain.** A protection domain is a segment, a virtual page, a BAT area, or a range of unmapped effective addresses. It is defined only when the appropriate relocate bit in the MSR (IR or DR) is 1.

## Q

**Quad word.** A group of 16 contiguous locations starting at an address divisible by 16.

**Quiesce.** To come to rest. The processor is said to quiesce when an exception is taken or a **sync** instruction is executed. The instruction stream is stopped at the decode stage and executing instructions are allowed to complete to create a controlled context for instructions that may be affected by out-of-order, parallel execution. See *Context synchronization*.

## R

**rA.** The **r**A instruction field is used to specify a GPR to be used as a source or destination.

**rB.** The **r**B instruction field is used to specify a GPR to be used as a source.

**rD.** The **r**D instruction field is used to specify a GPR to be used as a destination.

**rS.** The **r**S instruction field is used to specify a GPR to be used as a source.

**Record bit.** Bit 31 (or the Rc bit) in the instruction encoding. When it is set, updates the condition register (CR) to reflect the result of the operation.

**Reconciliation sublayer.** Sublayer that maps the terminology and commands used in the MAC layer into electrical formats appropriate for the physical layer entities.

**Reduced instruction set computing (RISC).** An *architecture* characterized by fixed-length instructions with nonoverlapping functionality and by a separate set of load and store instructions that perform memory accesses.

**Referenced bit.** One of two *page history bits* found in each *page table entry*. The processor sets the *referenced bit* whenever the page is accessed for a read or write. See also *Page access history bits*.

**Reservation.** The processor establishes a reservation on a *cache block* of memory space when it executes an **lwarx** instruction to read a memory semaphore into a GPR.

**Reservation station.** A buffer between the dispatch and execute stages that allows instructions to be dispatched even though the results of instructions on which the dispatched instruction may depend are not available.

## S

**Secondary cache.** A cache memory that is typically larger and has a longer access time than the primary cache. A secondary cache may be shared by multiple devices. Also referred to as L2, or level-2, cache.

**Set** (*v*). To write a nonzero value to a bit or bit field; the opposite of *clear*. The term ‘set’ may also be used to generally describe the updating of a bit or bit field.

**Set** (*n*). A subdivision of a *cache*. Cacheable data can be stored in a given location in one of the sets, typically corresponding to its lower-order address bits. Because several memory locations can map to the same location, cached data is typically placed in the set whose *cache block* corresponding to that address was used least recently. See *Set-associative*.

**Set-associative.** Aspect of cache organization in which the cache space is divided into sections, called *sets*. The cache controller associates a particular main memory address with the contents of a particular set, or region, within the cache.

**Slave.** The device addressed by a master device. The slave is identified in the address tenure and is responsible for supplying or latching the requested data for the master during the data tenure.

**Snooping.** Monitoring addresses driven by a bus master to detect the need for coherency actions.

**Snoop push.** Response to a snooped transaction that hits a modified cache block. The cache block is written to memory and made available to the snooping device.

**Stall.** An occurrence when an instruction cannot proceed to the next stage.

**Sticky bit.** A bit that when *set* must be cleared explicitly.

**Superscalar machine.** A machine that can issue multiple instructions concurrently from a conventional linear instruction stream.

**Supervisor mode.** The privileged operation state of a processor. In supervisor mode, software, typically the operating system, can access all control registers and can access the supervisor memory space, among other privileged operations.

**Synchronization.** A process to ensure that operations occur strictly *in order*. See *Context synchronization*.

**System memory.** The physical memory available to a processor.

## T

**Tenure.** The period of bus mastership. There can be separate address bus tenures and data bus tenures.

**Throughput.** The measure of the number of instructions that are processed per clock cycle.

**Time-division multiplex (TDM).** A single serial channel used by several channels taking turns.

**Transaction.** A complete exchange between two bus devices. A transaction is typically comprised of an address tenure and one or more data tenures, which may overlap or occur separately from the address tenure. A transaction may be minimally comprised of an address tenure only.

**Transfer termination.** Signal that refers to both signals that acknowledge the transfer of individual beats (of both single-beat transfer and individual beats of a burst transfer) and to signals that mark the end of the tenure.

**Translation lookaside buffer (TLB).** A cache that holds recently-used *page table entries*.

## U

**User mode.** The operating state of a processor used typically by application software. In user mode, software can access only certain control registers and can access only user memory space. No privileged operations can be performed. Also referred to as problem state.

## V

**Virtual address.** An intermediate address used in the translation of an *effective address* to a physical address.

**Virtual memory.** The address space created using the memory management facilities of the processor. Program access to *virtual memory* is possible only when it coincides with *physical memory*.

## W

**Way.** A location in the cache that holds a cache block, its tags, and status bits.

**Word.** A 32-bit data element.

**Write-back.** A cache memory update policy in which processor write cycles are directly written only to the cache. External memory is updated only indirectly, for example, when a modified cache block is *cast out* to make room for newer data.

**Write-through.** A cache memory update policy in which all processor write cycles are written to both the cache and memory.

# Index

## A

- Address maps
  - addressing on PCI/PCI-X bus, 16-47
- Address mask (LBC), 13-13
- Address translation and mapping units (ATMUs)
  - inbound windows, 2-10
    - illegal interactions between inbound ATMUs and local access windows, 2-10
    - PCI Express, 17-25
      - endpoint (EP) mode, 17-25
      - root complex (RC) mode, 17-25
    - PCI/PCI-X—4 windows, 2-10, 16-19
  - local access windows, 2-3–2-10
    - *see also* Local access windows
  - outbound windows, 2-9
    - PCI Express, 17-20
    - PCI/PCI-X—4 windows, 16-15
- Addressing
  - PCI bus addressing, 16-48
    - configuration space, 16-48
    - I/O space, 16-48
    - memory space, 16-47
- AFEU
  - context, 10-96
  - context memory, 10-96, 10-97
  - data size register, 10-90
  - FIFOs, 10-97
  - interrupt control register, 10-94
  - interrupt status register, 10-92
  - key registers, 10-97
  - key size register, 10-89
  - mode register, 10-89
  - reset control register, 10-91
  - status register, 10-62, 10-91, 10-138, 10-157
- Alignment, byte (PCI/PCI-X), 16-49
- Arbitration
  - I$^2$C interface
    - arbitration control, 11-15
    - loss of arbitration—forcing of slave mode, 11-24
    - procedure for arbitration, 11-15
  - PCI/PCI-X interface, 16-5, 16-42
- ARC Four execution unit (AFEU), 10-88
- Architecture
  - overview, 1-3
- ASLEEP (global utilities asleep) signal, 23-2
- ATMUs, *see* Address translation and mapping units

## B

- Battery backup, self-refresh mode, 8-94
- Big-endian, 21-6
- Block diagrams
  - DDR controller, 8-2, 8-60
  - debug modes, watchpoint monitor, and trace buffer, 25-1
  - DMA controller, 15-1
  - DUART, 12-2
  - e500 coherency module (ECM), 7-1
  - eTSEC, 14-2
  - GPIO*n* module, 22-1
  - I$^2$C interface, 11-1
  - interrupt controller (PIC), 9-1
  - L2 cache/SRAM, 6-1
  - local bus controller (LBC), 13-1
  - PCI Express, 17-2
  - PCI/PCI-X controller, 16-1
  - performance monitor, 24-2
- Boot mode
  - CPU holdoff (POR), 4-17, 7-4
  - POR status register (PORBMSR), 23-6
- Boot page translation, 4-7
- Boot ROM location (POR), 4-14
- Boot sequencer
  - boot holdoff mode (POR), 4-18, 7-4
  - boot page translation, 4-7
  - I$^2$C interface, 11-2, 11-17–11-20
  - overview, 4-8
  - POR configuration, 4-18
- Buffer descriptors, *see* eTSEC, buffer descriptors
- Burst operations (PCI)
  - *see* PCI/PCI-X controller, bus protocol
- Bus operations
  - PCI/PCI-X, *see* PCI/PCI-X controller, bus protocol
- Byte alignment (PCI/PCI-X), 16-49

## C

- Chaining
  - performance monitor events, 24-27
- CKSTP_IN (global utilities checkstop in) signal, 23-2
- CKSTP_OUT (global utilities checkstop out) signal, 23-3
- CLK_OUT (global utilities clock out) signal, 23-3, 23-33
- Clocks
  - DDR clock distribution, 8-76
  - DDR controller clock disable, 23-32, 23-33

device clock signals summary, 4-3
see also Signals, clock
device clocking operation, 4-24–4-27
CCB (platform) clock, 4-24
minimum CCB frequency equations, 4-26
Ethernet clocks, 4-26
PCI Express clocks, 4-25
system clock/PCI clock, 4-24
eTSEC
inputs and outputs, 14-10
management clock out (EC_MDC), 14-11, 14-81
$I^2C$
clock stretching, 11-17
clock synchronization, 11-16
input synchronization and digital filter, 11-16
LBC bus clocks and clock ratios, 13-4
clock ratio register (LCRR), 13-34
PCI/PCI-X clocking, 16-45, 16-50
POR settings
e500 core PLL ratio, 4-12
system/CCB PLL ratio, 4-11
Coherency rules
L2 cache, 6-28
Commands
PCI, *see* PCI/PCI-X controller
Communications processor module (CPM)
serial peripheral interface (SPI)
multi-master operation, 18-3
Configuration
DDR, 8-12–8-51, 8-62
ECM
CCB address configuration register (EEBACR), 7-3
CCB port configuration register (EEBPCR), 7-4
eTSEC interfaces, 14-207–??
LBC
configuration register (LBCR), 13-32
PCI/PCI-X
configuration access registers, 16-60
configuration cycles, 16-58
configuration space header, 16-58
PIC
global configuration register, 9-21
POR, *see* Power-on-reset (POR)
Configuration space
PCI Express, 17-43
PCI/PCI-X addressing, 16-48
Configuration, control, and status
accessing CCSR memory from external masters, 2-11
accessing CCSRs, 4-4
alternate configuration space (ALTBAR and ALTCAR), 4-5
boot page translation, 4-7
CCSR memory map, 2-10–2-14
CCSRBAR update guidelines, 4-4
memory map/register definition, 4-3
organization of CCSR memory, 2-11
Context ID registers, 25-22–25-23
Controller registers, 10-50
Crypto-channel
configuration register, 10-37
Crypto-channel registers, 10-37
CTS, see DUART_CTS[0:1]

## D

Data cache
*see* L2 cache/SRAM
Data encryption standard execution units (DEU), 10-57, 10-108, 10-117
DDR controller
address signal mappings, 8-5
battery-backed RAM and forced self-refresh mode, 8-94
block diagram, 8-2, 8-60
clock distribution, 8-76
clocks
disabling, 23-32, 23-33
configuration, example, 8-62
data beat ordering, 8-83
debug mode
signal selection (POR), 4-23
source and target ID, 25-4, 25-24
driver impedance calibration, 8-9
error checking and correcting (ECC), 6-39, 8-84
testing ECC with error injection, 8-51–8-53
error handling, 8-53, 8-86
features, 8-2
functional description, 8-60
initialization/application information, 8-87
programming different memory types, 8-89
interrupts, 8-57
memory map/register definition, 8-10
modes of operation, 8-3
on-die termination for CSs, 8-9
page mode and logical bank retention, 8-83
performance monitor events, 24-17
register descriptions, 8-12
by acronym, *see* Register Index
configuration registers, 8-12–8-51
error handling registers, 8-53–8-60
error injection registers, 8-51–8-53
SDRAM operation, 8-64
address multiplexing, 8-66
initialization sequence, 8-94
JEDEC standard interface commands, 8-71
mode-set command timing, 8-76

    organizations supported, 8-64
    refresh operation, 8-79
      power-saving modes, 8-80
      timing, 8-80
    registered DIMM mode, 8-77
    timing, 8-72
    write timing adjustments, 8-78
  self-refresh
    operation in sleep mode, 8-82
    using forced mode for battery backup, 8-94
  signals summary, 8-3
    *see also* Signals, DDR
Debug modes
  and watchpoint monitor signals summary, 25-5
    see also Signals, debug
  and watchpoint monitor/trace buffer block diagram, 25-1
  DDR signal selection (POR)
    ECC pins used for debug, 4-23
  DDR source ID debug modes, 25-4, 25-24
    source ID on debug signals, 25-25
    source ID on ECC pins, 25-25
  DDR/LBC signal selection (POR), 4-23
  features, 25-3
  functional description, 25-24
  LBC source ID debug mode, 13-4, 25-4, 25-24
  memory map/register definition, 25-9
  modes of operation (set at POR), 25-3
  overview, 25-1
  PCI/PCI-X
    source ID debug mode, 25-24
  performance monitor events, 24-26
  POR status (global utilities), 23-12
  READY negation, 4-2
  software debug
    context ID registers, 25-22
  trace buffer, see Trace buffer
  watchpoint, see Watchpoint monitor
Descriptor structure, 10-20
DEU
  FIFOs, 10-117
  interrupt control register, 10-66, 10-115
  interrupt status register, 10-64, 10-113
  IV register, 10-117
  key registers, 10-117
  key size register, 10-61, 10-110
  mode register, 10-58, 10-109
  reset control register, 10-62, 10-111
DMA channel 2 and 3 signal select, 23-14
DMA controller
  block diagram, 15-1
  channel operation, 15-24
    bandwidth control, 15-31
    channel abort, 15-31
    channel state, 15-31
    stride size and distance, 15-32
  descriptor formats, 15-33
  error handling, 15-32
  features, 15-2
  functional description, 15-24
  interrupts, 15-9–15-12, 15-14, 15-19, 15-23, 15-32
  limitations and restrictions, 15-36
  memory map/register definition, 15-5
  modes of operation, 15-2
    basic mode transfer, 15-25
      basic chaining mode, 15-26
      basic chaining single-write start mode, 15-27
      basic direct mode, 15-25
      basic direct single-write start mode, 15-26
    channel continue mode for cascading transfer chains, 15-30
      basic channel continue mode, 15-30
      extended mode, 15-31
    extended DMA mode transfer, 15-27
      extended chaining mode, 15-28
      extended chaining single-write start mode, 15-28
      extended direct mode, 15-27
      extended direct single-write start mode, 15-28
    external control mode transfer, 15-29
  overview, 15-2
  performance monitor events, 24-18
  register descriptions, 15-8–15-24
    by acronym, see Register Index
  signal select—channel 2 and 3, 23-14
  signals summary, 15-4
    *see also* Signals, DMA controller
  system considerations, 15-38
    unusual scenarios, 15-39
      DMA to configuration and control registers, 15-39
      DMA to DUART, 15-39
      DMA to e500 core, 15-39
      DMA to I2C, 15-39
  transfer interfaces, 15-32
DMA engine, 21-40
DMA_DACK[0:3] (DMA acknowledge) signals, 15-5
DMA_DDONE[0:3] (DMA done) signals, 15-5
DMA_DREQ[0:3] (DMA request) signals, 15-5
Do-Complete-Split state
  asynchronous, 21-85
  periodic interrupt, 21-94
Do-Start-Split state
  asynchronous, 21-85
  periodic interrupt, 21-93
DUART
  asynchronous communication bits, 12-1

    - parity bit, 12-20
    - START bit, 12-20
    - STOP bit, 12-20
  - baud-rate generator logic, 12-20
  - block diagram, 12-2
  - divisor latch access bit (ULCRn[DLAB]), 12-3, 12-11
  - error handling, 12-21
    - framing error, 12-8, 12-15, 12-20, 12-21
    - overrun error, 12-22
    - parity error, 12-22
  - errors detected, 12-2
  - features, 12-1
  - functional description, 12-19
  - initialization/application information, 12-23
  - interrupts
    - interrupt control logic, 12-23
    - interrupt enable and control registers, 12-7–12-10
  - memory map/register definition, 12-3
  - modes of operation, 12-2
    - DMA mode selection, 12-22
    - FIFO mode, 12-22
      - interrupts, 12-22
    - local loopback mode, 12-21
  - overview, 12-1
  - PC16450 UART compatibility, 12-1
  - performance monitor events, 24-26
  - register descriptions, 12-5–12-19
    - UART0 register offsets, 12-3
    - UART1 register offsets, 12-3
  - serial interface data format, 12-2
  - serial interface operation, 12-19–12-20
    - data transfer, 12-20
    - START bit, 12-20
    - STOP bit, 12-20
    - transaction protocol example, 12-19
  - signals summary, 12-3
    - see also Signals, DUART

## E

- e500 coherency module (ECM)
  - block diagram, 7-1
  - CCB arbiter, 7-9
  - CCB interface, 7-10
  - configuration
    - CCB address configuration register (EEBACR), 7-3
    - CCB port configuration register (EEBPCR), 7-4
  - error handling
    - error handling registers, 7-6–7-9
  - features, 7-2
  - functional description, 7-9
  - global data multiplexor, 7-10
  - I/O arbiter, 7-9
  - initialization/application information, 7-10–7-11
  - interrupts
    - ECM error enable register (EEER), 7-7
  - memory map/register definition, 7-3
  - overview, 7-2
  - performance monitor events, 24-19
  - register descriptions, 7-3
    - by acronym, see Register Index
  - transaction queue, 7-10
- e500 core
  - boot mode (POR), 4-17
  - interrupts
    - sources, 9-4
  - registers
    - PVR (processor version register), 5-6
    - SVR (system version register), 23-30
    - TCR (timer control register), 5-5
  - time base
    - RTC (real time clock) signal options, 4-3, 4-26
- EC_GTX_CLK125 (eTSEC gigabit transmit 125 MHz source) signal, 14-11
- EC_MDC (eTSEC management data clock) signal, 14-11
- EC_MDIO (eTSEC management data input/output, BIDI) signal, 14-11
- Error handling
  - DDR, 8-53–8-60, 8-86
  - DMA, 15-32
  - DUART, 12-2, 12-21
    - framing error, 12-8, 12-15, 12-20, 12-21
    - overrun error, 12-22
    - parity error, 12-22
  - ECM
    - error handling registers, 7-6–7-9
  - eTSEC, 14-175–14-176
  - I$^2$C interface
    - boot sequencer mode, 11-19
  - L2 cache/SRAM
    - error handling registers, 6-17
    - error injection, 6-18
  - LBC
    - transfer error registers, 13-26–13-31
  - PCI Express registers, 17-30–17-43
  - PCI/PCI-X
    - address/data parity, 16-54, 16-65, 16-66
    - reporting, 16-65–16-66
      - $\overline{\text{PERR}}$ and $\overline{\text{SERR}}$ signals, 16-66
      - target-initiated termination, 16-53
    - retry transactions, 16-53
    - target-abort, 16-53
    - target-disconnect, 16-53
- eTSEC
  - block diagram, 14-2

buffer descriptors, 14-200–14-207
  receive buffer descriptors (RxBD), 14-205
  transmit buffer descriptors (TxBD), 14-202
clocks
  inputs and outputs, 14-10
  management clock out (EC_MDC), 14-11, 14-81
  operation, 4-26
configuration of interfaces, 14-207–??
  8-bit FIFO mode, 14-232
  GMII interface mode, 14-212
  MAC configuration, 14-71
  MII interface mode, 14-208
  RGMII interface mode, 14-220
  RMII interface mode, 14-224
  RTBI interface mode, 14-228
  TBI interface mode, 14-216
data width (POR), 4-20
error-handling, 14-175–14-176
eTSEC1 protocol (POR), 4-21
eTSEC3 protocol (POR), 4-21
features, 14-2
FIFO interface connections, 14-156
  8-bit encoded packet FIFO mode, 14-159
  8-bit GMII-style packet FIFO mode, 14-158
  CRC appending and checking, 14-157
  flow control, 14-157
  signal summary, 14-159
functional description, 14-146
gigabit Ethernet channel operation, 14-160
  flow control, 14-170
  frame reception, 14-163
  frame recognition, 14-166
  frame transmission, 14-162
  initialization sequence, 14-160
    soft reset and reconfiguring procedure, 14-161
  internal and external loop back, 14-175
  inter-packet gap time, 14-174
  Magic Packet mode, 14-170
  preamble customization, 14-164
  RMON support, 14-166
hash function
  algorithm, 14-168
  registers, 14-115–14-116
initialization/application information, 14-207–??
  gigabit Ethernet channel, 14-160
    soft reset and reconfiguring procedure, 14-161
  *see also* eTSEC, configuration
interrupts, 14-171–14-174
  interrupt coalescing, 14-172
    by frame count threshold, 14-172
    by timer threshold, 14-173
  interrupt registers, 14-27–14-33
lossless flow control, 14-191
  back pressure determination and free buffers, 14-191
  software use of hardware-initiated back pressure, 14-193
MAC functionality, 14-71–14-87
  configuration, 14-71
  CSMA/CD control, 14-71
  handling packet collisions, 14-72
  packet flow control, 14-72
  PHY links control, 14-73
  registers, 14-74–14-87
memory map/register definition, 14-14
  detailed memory map, 14-15–14-26
  top-level module map, 14-15
modes of operation, 14-5
  RMON support, 14-87
overview, 14-1
physical interface connections, 14-146
  gigabit media-independent interface (GMII), 14-148
  media-independent interface (MII), 14-146
  reduced gigabit media-independent interface (RGMII), 14-148
  reduced media-independent interface (RMII), 14-146
  reduced ten-bit interface (RTBI), 14-150
  ten-bit interface (TBI), 14-149
quality of service (QoS) support, 14-181–14-191
  receive queue filer, 14-183
  transmission scheduling, 14-189
register descriptions, 14-26–14-146
  by acronym, *see* Register Index
  DMA attribute registers, 14-118–14-120
  FIFO registers, 14-116–14-118
  general control and status registers, 14-26–14-40
  hash function registers, 14-115–14-116
  lossless flow control registers, 14-120–14-121
  MAC registers, 14-74–14-87
  MIB registers, 14-87–14-115
  receive control and status registers, 14-53–14-70
  ten-bit interface registers, 14-135–14-145
  transmit control and status registers, 14-40–14-52
signals, ??–14-13
  FIFO interface signal summary, 14-159
  *see also* Signals, eTSEC
  summary, 14-6
signals<$startmode, 14-9
TCP/IP off-load, 14-176–14-181
  frame control blocks, 14-177
  receive path off-load, 14-179
  transmit path off-load, 14-178
EU
  access, 10-49
  assignment status register, 10-50
External system configuration

POR (LAD[0:31]) status, 4-24, 23-13
External writes, *see* L2 cache/SRAM, stashing

F

Fetch register, 10-44
FIFO RAM controller, 21-41
FSTNs
host controller operational model, 21-89
software operational model, 21-91

G

General Purpose I/O module (GPIO), *see* GPIO
Global utilities
clock out
CLK_OUT signal, 23-3, 23-33
clock out control register (CLKOCR), 23-33
overview, 23-1
DDR controller
clock disable, 23-32, 23-33
DMA signal multiplex control register (PMUXCR), 23-14
features, 23-1
functional description, 23-47
interrupt and local bus signal multiplexing, 23-1, 23-14
LBC voltage select, 23-31
machine check summary
sources of *mcp* (MCPSUMR), 23-26
memory map/register definition, 23-3
overview, 23-1
POR configuration
boot mode status register (PORBMSR), 23-6
debug mode status register (PORDBGMSR), 23-12
device status register (PORDEVSR), 23-9
I/O impedance status register (PORIMPSCR), 23-8
LAD[0:31] external system configuration (GPPORCR), 4-24, 23-13
PLL status register (PORPLLSR), 23-5
*see also* Power-on reset (POR)
power management
block disable (DEVDISR), 23-16
device mode control and status register (POWMGTCSR), 23-19, 23-21
features, 23-1
functional description, 23-47
processor version register (PVR), 23-29
register descriptions, 23-5
by acronym, *see* Register Index
reset
HRESET_REQ control, 23-30
RapidIO and PCI Express reset requests (RSTRSCR), 23-28
signals summary, 23-2
*see also* Signals, global utilities
system version register (SVR), 23-30
GPCM (LBC general-purpose chip-select machine), 13-48
see also Local bus controller (LBC)
GPIO
block diagram, 22-1
features, 22-1
memory map/register definition, 22-2
overview, 22-1
registers, 22-2–22-5
signals, 22-2

H

Hash function, *see* eTSEC, hash function
HRESET (hard reset) signal, 4-2, 4-8
HRESET_REQ (hard reset request) signal, 4-2, 11-18, 11-19

I

I/O impedance
LBC and PCI/PCI-X signals
control and status register (global utilities), 23-8
PCI/PCI-X interface (POR), 4-22
I/O space
PCI/PCI-X addressing, 16-48
I$^2$C interface
arbitration
arbitration control, 11-15
loss of arbitration—forcing of slave mode, 11-24
procedure for arbitration, 11-15
block diagram, 11-1
boot sequencer
POR configuration, 4-18
boot sequencer mode, 11-2, 11-17–11-20
error condition behavior, 11-19
calling address match condition, 11-6
clock control, 11-16
clock stretching, 11-17
clock synchronization, 11-16
input synchronization and digital filter, 11-16
master mode, 11-16
slave mode, 11-16
data transfer, 11-13
error handling
boot sequencer mode, 11-19
features, 11-2
frequency divider
frequency divider register (I2CFDR), 11-6
functional description, 11-11
handshaking, 11-16
implementation details, 11-13
address compare, 11-15

control transfer, 11-14
transaction monitoring, 11-14
initialization/application information, 11-21–11-25
boot sequencer mode, see I$^2$C interface, boot sequencer mode
generation of SCL when SDA low, 11-23
initialization sequence, 11-21
post-transfer software response, 11-22
repeated START generation, 11-23
START generation, 11-12, 11-21
STOP generation, 11-13, 11-22
interrupts
calling address match condition, 11-6
flowchart for interrupt service routine, 11-24
interrupt after transfer, 11-22
interrupt enable bit (I2CCR[MIEN]), 11-8
interrupt on START, 11-22
interrupt pending status bit (I2CSR[MIF]), 11-10
interrupt-driven byte-to-byte transfers, 11-2
read of last byte, 11-22
slave mode interrupt service routine guidelines, 11-23
for slave transmitter routine, 11-23
loss of arbitration, 11-24
memory map/register definition, 11-4
modes of operation, 11-2
boot sequencer mode, 11-2, 11-17–11-20
interrupt-driven byte-to-byte data transfer, 11-2
master mode, 11-2
slave mode, 11-2
overview, 11-2
register descriptions, 11-5
by acronym, see Register Index
signals summary, 11-3
see also Signals, I$^2$C
transaction protocol, 11-11
handshaking, 11-16
repeated START condition, 11-3, 11-13
slave address transmission, 11-12
START condition, 11-3, 11-12, 11-21
STOP condition, 11-3, 11-13, 11-22
ID register, 10-54
Initialization
DDR (initialization and application information), 8-87
programming different memory types, 8-89
ECM (initialization and application information), 7-10–7-11
eTSEC (initialization and application information), 14-160, 14-207–??
*see also* eTSEC, configuration
I$^2$C interface (initialization and application information), 11-21–11-25
boot sequencer mode, see I$^2$C interface, boot sequencer mode
generation of SCL when SDA low, 11-23
initialization sequence, 11-21
post-transfer software response, 11-22
repeated START generation, 11-23
START generation, 11-12, 11-21
STOP generation, 11-13, 11-22
PCI/PCI-X (initialization and application information), 16-67–??
PIC (initialization and application information), 9-58
watchpoint monitor and trace buffer, 25-30
Initialization/application information, 18-14, 20-44
host controller, 21-62
LBC, 13-90–??
Initiator write, 10-48
Interfaces
USB, 21-2
Interrupt
clear register, 10-54
mask register, 10-50
status register, 10-53
Interrupt controller (PIC)
block diagram, 9-1
configuration (global), 9-21
critical interrupts, 9-6, 9-29, 9-31
destination (interrupt routing)
IRQ_OUT, 9-6
end of interrupt (EOI), 9-51, 9-54
external interrupts
routed to critical interrupt (cint), 9-31
routed to IRQ_OUT, 9-30
features, 9-2
flow (interrupt processing), 9-52
functional description, 9-52
global timers, 9-23, 9-57
cascading of timers, 9-27, 9-28
clocking of timers, 9-24, 9-28
RTC (real time clock) signal options, 4-3, 4-26, 9-27, 9-28
initialization/application information, 9-58
interprocessor interrupts, 9-55
interrupt acknowledge (IACK) signaling, 9-50, 9-54
interrupt routing (mixed mode), 9-6
interrupt source priorities, 9-54
memory map/register definition, 9-9
messaging interrupts, 9-30, 9-31, 9-56
modes of operation, 9-4, 9-21
mixed mode, 9-5
pass-through mode (to support external interrupt controllers), 9-5
nesting of interrupts, 9-55

- overview, 9-1
- performance monitor events, 24-21
- processor core interrupt sources, 9-4
  - critical interrupt (cint) sources, 9-29
- programming guidelines, 9-58
  - changing interrupt source configuration, 9-60
- register descriptions, 9-19
  - by acronym, see Register Index, 9-19
  - global registers, 9-19–9-23
  - global timer registers, 9-23–9-28
  - interrupt source configuration registers, 9-24–9-26, 9-40–9-46
  - message registers, 9-34–9-36
  - non-accessible registers
    - interrupt pending register (IPR), 9-52
    - interrupt request register (IRR), 9-53
  - per-CPU registers, 9-46–9-51
  - performance monitor mask registers, 9-32–9-34
  - summary registers, 9-28–9-32
- reset of PIC, 9-21, 9-58
- reset processor from software, 9-22, 9-58
- signals summary, 9-8
  - see also Signals, PIC
- simultaneous interrupts, priorities, 9-54
- sources of interrupts, 9-5
  - internal (to PIC) interrupt destinations, 9-29, 9-30, 9-31, 9-32
  - internal (to PIC) interrupt sources, 9-6
- spurious vector generation, 9-23, 9-55
- vendor identification, 9-21

Interrupts
- channel done, 10-35
- channel error, 10-35
- DDR, 8-57
- DMA, 15-9–15-12, 15-14, 15-19, 15-23, 15-32
- DUART
  - interrupt control logic, 12-23
  - interrupt enable and control registers, 12-7–12-10
- ECM interrupt register (ECM error enable register—EEER), 7-7
- eTSEC, 14-171–14-174
  - interrupt registers, 14-27–14-33
- general, 10-37
- I$^2$C interface
  - calling address match condition, 11-6
  - flowchart for interrupt service routine, 11-24
  - interrupt after transfer, 11-22
  - interrupt enable bit (I2CCR[MIEN]), 11-8
  - interrupt on START, 11-22
  - interrupt pending status bit (I2CSR[MIF]), 11-10
  - interrupt-driven byte-to-byte transfers, 11-2
  - read of last byte, 11-22
  - slave mode interrupt service routine guidelines, 11-23
    - for slave transmitter routine, 11-23
    - loss of arbitration, 11-24
- IRQ[9:11] signal select, 23-14
- LBC interrupt register, 13-29
- PCI/PCI-X error enable register, 16-26
- performance monitor (PIC), 24-21
- see also Interrupt controller (PIC)
- split transaction, 21-86

IRQ[0:11] (interrupt request 0–11) signals, 9-8

IRQ[9:11] signal select
- global utilities, 23-14

IRQ_OUT (interrupt request out) signal, 9-8, 9-28

Isochronous transfer descriptor (iTD), see USB interface, isochronous (high-speed) transfer descriptor (iTD)

## J

JTAG test access port
- signals summary, 25-5
  - see also Signals, JTAG, 25-5

## L

L2 cache/SRAM
- allocation of lines, 6-32
- block diagram, 6-1
- coherency rules, 6-28
- error handling registers, 6-17
- error injection, 6-18
- external writes, *see* stashing
- flash clearing, instruction and data locks, 6-31
- locking
  - clearing locks on selected lines, 6-30
  - entire, 6-29
  - programmed memory ranges, 6-30
  - selected lines, 6-30
  - with stale data, 6-31
- memory map/register definition, 6-8
- memory-mapped SRAM
  - coherency rules, 6-29
- memory-mapped windows, 2-4
- operation, 6-33
- overview, 6-1
- performance monitor events, 24-26
- PLRU bit update considerations, 6-32
- register descriptions, 6-10–6-25
- replacement policy, 6-31
- SRAM features, 6-2
- stashing, 6-25
- state transitions, 6-35
  - due to core-initiated transactions, 6-35
  - due to system-initiated transactions, 6-38

  timing, 6-27
LA[27:31] (LBC non-multiplexed address) signals, 13-7
LAD[0:31] (LBC multiplexed address/data) signals, 13-7
LALE (LBC external address latch enable) signal, 13-5, 13-43
LBCTL (LBC data buffer control) signal, 13-7, 13-46
LBS[0:3] (LBC UPM byte select) signals, 13-6
LCK[0:2] (LBC clock) signals, 13-8
$\overline{\text{LCS}}$[0:7] (LBC chip select) signals, 13-6
$\overline{\text{LCS}}$[5:7] signal select
  global utilities, 23-14
LCS0 (LBC chip select 0) signal, 13-58, 13-72
LDP[0:3] (LBC data parity) signals, 13-8
LGPL0 (LBC GP line 0) signal, 13-6
LGPL1 (LBC GP line 1) signal, 13-6
LGPL2 (LBC GP line 2) signal, 13-6
LGPL3 (LBC GP line 3) signal, 13-6
LGPL4 (LBC GP line 4) signal, 13-7
LGPL5 (LBC GP line 5) signal, 13-7
LGTA (LBC GPCM transfer acknowledge) signal, 13-7, 13-57
Little-endian, 21-6
Local access windows, 2-3–2-10
  ATMUs, *see* Address translation and mapping units (ATMUs)
  configuring local access windows, 2-8
  distinguishing local access windows from other mapping functions, 2-9
  illegal interactions
    between inbound ATMUs and local access windows, 2-10
    between local access windows and DDR SDRAM chip selects, 2-9
  L2 cache/SRAM window interactions, 2-4
  precedence if overlapping among themselves, 2-8
  precedence if overlapping with L2 cache/SRAM windows, 2-4
  registers, 2-7–2-8
    by acronym, *see* Register Index
Local address map, 1-4
  see also Local access windows, 1-4
Local bus controller (LBC)
  address and address space checking, 13-43
  address mask field—option registers, 13-13
  atomic bus operations, 13-46
  block diagram, 13-1
  boot chip-select operation, 13-58, 13-72
  bus monitor, 13-47
  bus turnaround, 13-92
    additional address phases (UPM cycles), 13-93
    address following read, 13-93
    read data following address, 13-93
    read-modify-write cycle (parity), 13-93
  clocks and clock ratios, 13-4
    clock ratio register (LCRR), 13-34
  configuration
    LBC configuration register (LBCR), 13-32
  debug mode
    signal selection (POR), 4-23
    source and target ID, 25-4, 25-24
  error handling
    transfer error registers, 13-26–13-31
  external access termination (LGTA), 13-57
  features, 13-2
  functional description, 13-42
  general-purpose chip-select machine (GPCM), 13-48
    chip-select and write enable negation timing, 13-53
    chip-select assertion timing, 13-52
    extended hold time on read accesses, 13-56
    GPCM mode
      registers, 13-14, 13-16
    output enable timing, 13-56
    programmable wait state configuration, 13-53
    relaxed timing, 13-54
    timing configuration, 13-49, 13-51, 13-68
  initialization/application information, 13-90
  interrupts
    transfer error interrupt enable register (LTEIR), 13-29
  $\overline{\text{LCS}}$[5:7] signal select, 23-14
  memory map/register definition, 13-9
  memory refresh timer prescaler, 13-23
  modes of operation, 13-3
    bus clock and clock ratios, 13-4
    GPCM mode, registers, 13-14, 13-16
    source ID debug mode, 13-4
    UPM mode, registers, 13-19
  overview, 13-2
  performance monitor events, 24-26
  peripherals, 13-90
    GPCM timing, 13-92
    hierarchy for very high speeds, 13-91
    multiplexed address/data, 13-90
  port sizes, 13-93
  register descriptions, 13-10
    by acronym, see Register Index
  signals, 13-4
    see also Signals, LBC
  UPM interfaces, 13-74–13-104
    block diagram, 13-75
    extended hold time (reads), 13-90
    programming the UPMs, 13-78
    RAM array, 13-80
      address multiplexing, 13-86
      byte select signal timing, 13-84

chip select signal timing, 13-84
data timing, 13-87
general purpose signal timing, 13-85
LGPL[0:5] timing (LAST), 13-88
loop control, 13-85
RAM word definition, 13-80
REDO, 13-86
wait mechanism (WAEN), 13-88
signal timing, 13-80
synchronous UPWAIT (early transfer acknowledge), 13-89
UPM mode
registers, 13-19, 13-20
UPM requests, 13-75
exception requests, 13-77
memory access requests, 13-76
refresh timer requests, 13-77
software requests, 13-77
voltage selection, 23-31
ZBT SRAM interface, 13-99, 13-104
LOE (LBC GPCM output enable) signal, 13-6
LPBSE (LBC parity byte select) signal, 13-7
LSYNC_IN (LBC DLL synchronization in) signal, 13-8
LSYNC_OUT (LBC DLL synchronization out) signal, 13-8
LWE[0:3] (LBC GPCM write enable) signals, 13-6

## M

MA[0:14] (DDR address bus) signals, 8-7
MAC functionality, *see* eTSEC, MAC functionality
Machine check
MCP (processor machine check) signal, 9-8
*mcp* summary register (MCPSUMR), 23-26
SRESET (soft reset) signal, 4-8
Master control register, 10-55
MBA[0:1] (DDR logical bank address) signals, 8-7
MCAS (DDR column address strobe) signal, 8-8
MCK[0:5] (DDR clock output complement) signals, 8-10
MCK[0:5] (DDR clock output) signals, 8-10
MCKE[0:3] (DDR clock enable) signals, 8-10
MCP (processor machine check) signal, 9-8
MCS[0:3] (DDR chip select) signals, 8-8
MDEU
context registers, 10-143
data size register, 10-137, 10-142
FIFOs, 10-146
interrupt control register, 10-141
interrupt status register, 10-137, 10-139
key registers, 10-146
key size register, 10-136
mode register, 10-98, 10-132
reset control register, 10-137
MDIC[0:1] (DDR driver impedance calibration) signals, 8-9
MDM[0:8] (DDR SDRAM data output mask) signals, 8-9
MDQ[0:8] (DDR data bus strobe) signals, 8-6, 8-61
MDVAL (DDR/LBC debug mode data valid) signal, 4-23, 13-8, 25-3, 25-6
MECC[0:5] (DDR error correcting code) signals as debug, 25-3, 25-7
MECC[0:7] (DDR error correcting code) signals, 4-23, 8-7
Memory maps
CCSR memory, 2-4
accessing CCSR memory from external masters, 2-11
CCSR map, complete list of memory-mapped registers (by offset), 2-14
CCSR organization, 2-11
CCSR registers, 2-10–2-14
device-specific utilities, 2-13
general utilities registers, 2-12
programmable interrupt controller (PIC) space, 2-13
configuration, control, and status registers, 4-3
DDR controller, 8-10
illegal interaction between local access windows and DDR SDRAM chip selects, 2-9
debug, watchpoint, and trace buffer registers, 25-9
device memory map
address translation and mapping, 2-3
overview and example, 2-1
DMA, 15-5
DUART, 12-3
ECM, 7-3
eTSEC, 14-14
global utilities, 23-3
GPIO, 22-2
I$^2$C, 11-4
interrupt controller (PIC), 9-9
L2 cache/SRAM, 6-8
LBC, 13-9
PCI/PCI-X, 16-11
performance monitor, 24-3
USB interface, 21-4–21-5, ??–A-19
Memory space
PCI/PCI-X addressing, 16-47
Message digest execution unit (MDEU), 10-132
Message interrupts, see Interrupt controller (PIC), message interrupts
MODT[0:3] (DDR on-die termination) signals, 8-9
MRAS (DDR row address strobe) signal, 8-8
MSRCID[0:4] (DDR/LBC debug source ID) signals, 4-23, 13-8, 25-3, 25-7
MWE (DDR write enable) signal, 8-8, 8-9

## P

*PCI Express Base Specification, Rev. 1.0a*
*see* PCI Express controller

- PCI Express controller
  - accessing configuration space
    - endpoint (EP) mode, 17-44
    - root complex (RC) mode, 17-43
  - address translation and mapping unit (ATMU)
    - inbound windows, 17-25
      - endpoint (EP) mode, 17-25
      - root complex (RC) mode, 17-25
    - outbound windows, 17-20
  - block diagram, 17-2
  - clocks
    - minimum CCB frequency equation, 4-26
    - operation, 4-25
  - commands
    - command register, 17-46
  - configuration space accesses, 17-43
  - error handling registers, 17-30–17-43
  - features, 17-3
  - latency timer, 17-50
  - modes of operation, 17-4
    - link width, 17-4
    - root complex or endpoint mode, 17-4
  - overview, 17-1
  - POR configuration, 17-4
  - power management, 17-13–17-18, 17-69–17-70
  - register descriptions
    - configuration header registers, 17-45–17-67
      - 32-bit memory base address register, 17-52
      - 64-bit high memory base address register, 17-53
      - 64-bit low memory base address register, 17-53
      - base address registers, 17-51–17-53, 17-58
      - bridge control register, 17-67
      - bus status register, 17-47
      - cache line size register, 17-49
      - capabilities pointer register, 17-55, 17-65
      - command register, 17-46
      - configuration and status register base address (PCSRBAR), 17-51, 17-58
      - device ID register, 17-45, 17-54
      - I/O base register, 17-60
      - I/O base upper 16 bits register, 17-64
      - I/O limit register, 17-60
      - I/O limit upper 16 bits register, 17-65
      - interrupt line register, 17-55, 17-66
      - interrupt pin register, 17-56, 17-66
      - latency timer register, 17-50, 17-60
      - maximum latency (EP-mode) register, 17-57
      - memory base register, 17-62
      - memory limit register, 17-62
      - minimum grant (EP-mode) register, 17-56
      - prefetchable base upper 32 bits register, 17-64
      - prefetchable limit upper 32 bits register, 17-64
      - prefetchable memory base register, 17-63
      - prefetchable memory limit register, 17-63
      - primary bus number register, 17-58
      - revision ID register, 17-48
      - secondary bus number register, 17-59
      - secondary status register, 17-61
      - subordinate bus number register, 17-59
      - subsystem vendor ID register, 17-54
      - vendor ID register, 17-45
    - device-specific configuration space registers, 17-68–17-81
      - capabilities register, 17-71
      - capability ID register, 17-71
      - data capabilities register, 17-72
      - device control register, 17-72
      - device status register, 17-73
      - link capabilities register, 17-74
      - link control register, 17-74
      - link status register, 17-75
      - MSI message address register, 17-80
      - MSI message capability ID register, 17-79
      - MSI message control register, 17-80
      - MSI message data register, 17-81
      - MSI message upper address register, 17-81
      - power management capabilities register, 17-69
      - power management capability ID register, 17-69
      - power management data register, 17-70
      - power management status and control register, 17-70
      - root control register, 17-78
      - root status register, 17-78
      - slot capabilities register, 17-76
      - slot control register, 17-77
      - slot status register, 17-77
    - extended configuration space registers, 17-82–17-90
      - advanced error capabilities and control register, 17-87
      - advanced error reporting capability ID register, 17-83
      - correctable error mask register, 17-86
      - correctable error source ID register, 17-90
      - correctable error status register, 17-86
      - error source ID register, 17-90
      - header log register, 17-88
      - root error command register, 17-89
      - root error status register, 17-89
      - uncorrectable error mask register, 17-84
      - uncorrectable error severity register, 17-85
      - uncorrectable error status register, 17-83
    - memory-mapped registers, 17-6
      - ATMU registers, 17-20–17-30
      - by acronym, *see* Register Index
      - configuration access registers, 17-10–17-13, 17-43–17-45
      - error management registers, 17-30–17-43

    - IP block revision registers, 17-19
    - pwr mgmt and message registers, 17-13–17-18
  - signals summary, 17-4
    - *see also* Signals, PCI Express
- *PCI Local Bus Specification* configuration registers
  - *see* PCI/PCI-X controller, registers
- PCI/PCI-X controller
  - 64-bit/32-bit bus, 16-5
  - address bus decoding, 16-47
  - address translation and mapping unit (ATMU)
    - inbound windows (4), 2-10, 16-19
    - outbound windows (4), 16-15
  - arbiter configuration (POR), 4-23, 16-5
  - block diagram, 16-1
  - burst operations
    - cache wrap mode, 16-47
    - linear incrementing, 16-47
  - bus arbitration, 16-5, 16-42
  - bus protocol, 16-45
    - burst operation, 16-45
    - command encodings, 16-46
  - clocking, 16-45, 16-50
  - commands
    - command register, 16-31, 16-59
    - encodings, 16-46
    - interrupt-acknowledge transactions, 16-63
    - special-cycle, 16-64
  - configuration cycles, 16-58
  - configuration space addressing, 16-48
  - data bus width (POR), 4-22
  - debug mode
    - source and target ID (PCI_AD[63:59]), 25-24
  - error handling, 16-65–16-66
    - address/data parity, 16-54, 16-65, 16-66
    - detection and reporting, 16-65
    - reporting
      - $\overline{\text{PERR}}$ and $\overline{\text{SERR}}$ signals, 16-66
      - target-initiated termination, 16-53
    - retry transactions, 16-53
    - target-abort, 16-53
    - target-disconnect, 16-53
  - features, 16-4
  - functional description, 16-42
  - I/O impedance (POR), 4-22
  - I/O space addressing, 16-48
  - initialization/application information, 16-67–??
  - interrupts
    - error enable register, 16-26
  - latency timer, 16-36, 16-54, 16-59
  - memory map/register definition, 16-11
  - memory space addressing, 16-47
  - modes of operation, 16-4
    - agent configuration lock mode, 16-68
    - agent mode, 16-68
    - cache wrap mode, 16-47
    - host mode, 16-68
    - linear incrementing, 16-47
    - PCI-X mode
      - selection (POR), 4-22
  - overview, 16-2
  - performance monitor events
    - common events, 24-21
  - POR configuration, 16-67
  - power management
    - special-cycle operations, 16-64
  - register descriptions
    - configuration header registers, 16-30–??, 16-59
      - 32-bit memory base address register, 16-37
      - 64-bit high memory base address register, 16-38
      - 64-bit low memory base address register, 16-37
      - arbiter configuration register (PBACR), 16-41
      - base address registers, 16-36–16-38
      - base class code register, 16-35
      - bus function register (PBFR), 16-41
      - bus status register, 16-32, 16-49, 16-53, 16-65, 16-66
      - cache line size register, 16-35
      - capabilities pointer register, 16-39
      - command register, 16-31, 16-59
      - configuration and status register base address (PCSRBAR), 16-36
      - device ID register, 16-31, 16-39
      - interrupt line register, 16-39
      - interrupt pin register, 16-40
      - latency timer register, 16-36
      - maximum grant (MAX GNT) register, 16-40
      - maximum latency (MAX LAT) register, 16-41
      - programming interface register, 16-34
      - revision ID register, 16-34
      - subclass code register, 16-35
      - vendor ID register, 16-30, 16-38
    - memory-mapped registers, 16-11
      - ATMU inbound registers, 16-19–16-23
      - ATMU outbound registers, 16-15–16-19
      - by acronym, *see* Register Index
      - configuration access registers, 16-14–16-15, 16-60
      - error management registers, 16-23–16-29
  - signals summary, 16-6
    - *see also* Signals, PCI/PCI-X
  - target-abort termination, 16-53
  - target-disconnect cycles, 16-3, 16-53
  - target-initiated termination
    - target-abort error, 16-53
    - target-disconnect, 16-3, 16-53
  - transactions

    fast back-to-back transactions, 16-56
    interrupt-acknowledge transactions, 16-63
    read transactions, 16-50
    retry transactions, 16-53
    special-cycle transactions, 16-64
    timing diagrams, 16-50
    transaction termination, 16-52
      bus status register, termination status, 16-54
      completion, 16-53
      master-abort termination, 16-53
      master-initiated, 16-52
      target-initiated, 16-53, 16-54
      timeout, 16-53
    write transactions, 16-50, 16-51
  turnaround cycle, 16-49
PCI_AD[63:0] (PCI address/data bus) signals, 16-6
PCI_C/BE[7:0] (PCI command/byte enable) signals, 16-7, 16-46, 16-48, 16-49, 16-65
PCI_DEVSEL (PCI device select) signal, 16-7, 16-48
PCI_FRAME (PCI frame) signal, 16-7, 16-45
PCI_GNT[4:0] (PCI bus grant) signals, 16-8, 16-42
PCI_IDSEL (PCI initialization device) signal, 16-8
PCI_IRDY (PCI intitiator ready) signal, 16-8, 16-45
PCI_PAR (PCI parity) signal, 16-9
PCI_PERR (PCI parity error) signal, 16-9, 16-66
PCI_REQ[4:0] (PCI bus request) signals, 16-9, 16-42
PCI_SERR (PCI system error) signal, 16-10, 16-66
PCI_STOP (PCI stop) signal, 16-10, 16-49
PCI_TRDY (PCI target ready) signal, 16-10, 16-45, 16-52
Performance monitor (device)
  block diagram, 24-2
  burstiness, 24-14, 24-28
  control registers, 24-5–24-10
  counters (PMCn)
    chaining, 24-13
    registers, 24-10
    triggering, 24-13
  event counting, 24-12
  events, 24-16–24-27, ??–24-27
    chaining, 24-27
    DDR controller, 24-17
    debug, 24-26
    DMA controller, 24-18
    DUART, 24-26
    e500 coherency module (ECM), 24-19
    interrupt controller (PIC), 24-21
    L2 cache/SRAM, 24-26
    local bus controller (LBC), 24-26
    PCI/PCI-X common events, 24-21
  events triggered by watchpoint monitor, 25-26
  examples, 24-27
    burstiness event, 24-14
    burstiness event counting, 24-28
    simple event counting, 24-27, 24-28
    threshold event counting, 24-28
    triggering event counting, 24-27, 24-28
  external signals, 24-3
  features, 24-3
  functional description, 24-11
  interrupts, 24-11
  interrupts (from PIC) to generate events, 9-32
  masking interrupts (from PIC), 9-32
  memory map/register definition, 24-3
  overflow indication on TRIG_OUT, 25-24
  overview, 24-2
  threshold events, 24-12
Phase-locked loops (PLLs)
  POR status (global utilities), 23-5
PHY clocks, 21-4
PHY interface, 21-41
PKEU
  data size register, 10-149
  EU_GO register, 10-96, 10-116, 10-143, 10-154, 10-160
  interrupt control register, 10-153
  interrupt status register, 10-151
  key size register, 10-147, 10-148
  mode register, 10-147
  parameter memory A, 10-154
  parameter memory B, 10-155
  parameter memory E, 10-155
  parameter memory N, 10-155
  reset control register, 10-149
  status register, 10-112, 10-150
Power management
  block disable
    block disable control (DEVDISR), 23-16
  DDR interface, 8-80
  device low-power modes, 23-47–??
    control and status register (POWMGTCSR), 23-19, 23-21
    READY negation, 4-2
  PCI Express, 17-13–17-18, 17-69–17-70
  PCI special-cycle operations, 16-64
  *see also* Global utilities, power management
Power-on reset (POR)
  configuration
    boot ROM location, 4-14
    boot sequencer configuration, 4-18
    clock
      e500 core PLL ratio, 4-12
      system/CCB PLL ratio, 4-11
    CPU boot configuration, 4-17
    DDR debug mode (ECC pins used for debug), 4-23, 25-3
    eTSEC1 protocol, 4-21

eTSEC1–2 data width, 4-20
eTSEC3 protocol, 4-21
general-purpose (external system)
configuration—LAD[0:31] (GPPORCR), 4-24
memory debug select (DDR or LBC), 4-23, 25-3
PCI data bus width, 4-22
PCI debug configuration, 25-3
PCI I/O impedance, 4-22
PCI/PCI-X arbiter configuration, 4-23
PCI/PCI-X, modes of operation, 16-67
PCI-X mode selection, 4-22
configuration reporting
global utilities, 23-5, 23-6, 23-8, 23-9, 23-12, 23-13
debug modes summary, 25-3
hard reset, 4-8
output signal states during reset, 3-17
PCI Express, modes of operation, 17-4
reset configuration signals, 3-15
sequence of events, 4-9
and READY signal, 4-2, 4-10
Processor version (PVR), 23-29
Protocols
PCI, *see* PCI/PCI-X contoller, bus protocol
PVR (processor version register), see e500 core, registers

## Q

Quality of service (QoS), *see* eTSEC
Queue element transfer descriptor (qTD), see USB interface, queue element transfer descriptor (qTD)
Queue heads, see USB interface, queue heads

## R

Random number generator (RNG)
overview, 10-155
RapidIO controller
clocks
minimum CCB frequency equation, 4-26
READY signal, 4-2, 4-10, 25-23, 25-24
Registers
AFEU
data size, 10-90
interrupt control, 10-94
interrupt status, 10-92
key, 10-97
key size, 10-89
mode, 10-89
reset control, 10-91
status, 10-62, 10-91, 10-138, 10-157
by acronym (memory-mapped registers)
see Register Index
configuration, control, and status, 2-10–2-14, 4-3
device-specific utilities, 2-13
general utilities, 2-12
programmable interrupt controller (PIC) space, 2-13
context ID, 25-22–25-23
crypto-channel
configuration, 10-37
general, 10-37
DDR
configuration registers, 8-12–8-51
error handling registers, 8-53–8-60
error injection registers, 8-51–8-53
DEU
interrupt control, 10-66, 10-115
interrupt status, 10-64, 10-113
IV, 10-117
key, 10-117
key size, 10-61, 10-110
mode, 10-58, 10-109
reset control, 10-62, 10-111
ECM, 7-3
eTSEC, 14-26–14-146
DMA attribute registers, 14-118–14-120
FIFO registers, 14-116–14-118
general control and status registers, 14-26–14-40
hash function registers, 14-115–14-116
lossless flow control registers, 14-120–14-121
MAC registers, 14-74–14-87
MIB registers, 14-87–14-115
receive control and status registers, 14-53–14-70
ten-bit interface registers, 14-135–14-145
transmit control and status registers, 14-40–14-52
EU assignment status, 10-50
fetch, 10-44
global utilities, 23-5
POR boot mode status, 23-6
POR debug mode status, 23-12
POR device status, 23-9
POR external system configuration, 23-13
POR I/O impedance status, 23-8
POR PLL status, 23-5
GPIO, 22-2–22-5
I$^2$C interface, 11-5
ID, 10-54
interrupt
clear, 10-54
mask, 10-50
status, 10-53
L2 cache/SRAM registers, 6-8–6-25
LBC, 13-10
local access window registers
attributes registers (LAWAR0–LAWAR7), 2-7
base address registers (LAWBAR0–LAWBAR7), 2-7

master control, 10-55
MDEU
  context registers, 10-143
  data size, 10-137, 10-142
  interrupt control, 10-141
  interrupt status, 10-137, 10-139
  key, 10-146
  key size, 10-136
  mode, 10-98, 10-132
  reset control, 10-137
PCI Express
  configuration header registers, 17-45–17-67
    32-bit memory base address register, 17-52
    64-bit high memory base address register, 17-53
    64-bit low memory base address register, 17-53
    base address registers, 17-51–17-53, 17-58
    bridge control register, 17-67
    bus status register, 17-47
    cache line size register, 17-49
    capabilities pointer register, 17-55, 17-65
    command register, 17-46
    configuration and status register base address (PCSRBAR), 17-51, 17-58
    device ID register, 17-45, 17-54
    I/O base register, 17-60
    I/O base upper 16 bits register, 17-64
    I/O limit register, 17-60
    I/O limit upper 16 bits register, 17-65
    interrupt line register, 17-55, 17-66
    interrupt pin register, 17-56, 17-66
    latency timer register, 17-50, 17-60
    maximum latency (EP-mode) register, 17-57
    memory base register, 17-62
    memory limit register, 17-62
    minimum grant (EP-mode) register, 17-56
    prefetchable base upper 32 bits register, 17-64
    prefetchable limit upper 32 bits register, 17-64
    prefetchable memory base register, 17-63
    prefetchable memory limit register, 17-63
    primary bus number, 17-58
    revision ID register, 17-48
    secondary bus number, 17-59
    secondary status register, 17-61
    subordinate bus number, 17-59
    subsystem vendor ID, 17-54
    vendor ID, 17-45
  device-specific configuration space registers, 17-68–17-81
    capabilities register, 17-71
    capability ID register, 17-71
    device capabilities register, 17-72
    device control register, 17-72
    device status register, 17-73
    link capabilities register, 17-74
    link control register, 17-74
    link status register, 17-75
    MSI message address register, 17-80
    MSI message capability ID register, 17-79
    MSI message control register, 17-80
    MSI message data register, 17-81
    MSI message upper address register, 17-81
    power management capabilities register, 17-69
    power management capability ID register, 17-69
    power management data register, 17-70
    power management status and control register, 17-70
    root control register, 17-78
    root status register, 17-78
    slot capabilities register, 17-76
    slot control register, 17-77
    slot status register, 17-77
  extended configuration space registers, 17-82–17-90
    advanced error capabilities and control register, 17-87
    advanced error reporting capability ID register, 17-83
    correctable error mask register, 17-86
    correctable error source ID register, 17-90
    correctable error status register, 17-86
    error source ID register, 17-90
    header log register, 17-88
    root error command register, 17-89
    root error status register, 17-89
    uncorrectable error mask register, 17-84
    uncorrectable error severity register, 17-85
    uncorrectable error status register, 17-83
  memory-mapped registers
    ATMU registers, 17-20–17-30
    configuration access registers, 17-10–17-13, 17-43–17-45
    error management registers, 17-30–17-43
    IP block revision registers, 17-19
    pwr mgmt and message registers, 17-13–17-18
PCI/PCI-X
  configuration header registers, 16-30–??, 16-59
    32-bit memory base address register, 16-37
    64-bit high memory base address register, 16-38
    64-bit low memory base address register, 16-37
    arbiter configuration register (PBACR), 16-41
    base address registers, 16-36–16-38
    base class code register, 16-35
    bus function register (PBFR), 16-41
    bus status register, 16-32, 16-49, 16-53, 16-65, 16-66
    cache line size register, 16-35
    capabilities pointer register, 16-39
    command register, 16-31, 16-59

configuration and status register base address (PCSRBAR), 16-36
device ID register, 16-31, 16-39
interrupt line register, 16-39
interrupt pin register, 16-40
latency timer register, 16-36
maximum grant (MAX GNT) register, 16-40
maximum latency (MAX LAT) register, 16-41
programming interface register, 16-34
revision ID register, 16-34
subclass code register, 16-35
vendor ID, 16-30, 16-38
memory-mapped registers
ATMU inbound registers, 16-19–16-23
ATMU outbound registers, 16-15–16-19
configuration access registers, 16-14–16-15, 16-60
error management registers, 16-23–16-29
performance monitor, descriptions, 24-3
PIC, 9-19
global registers, 9-19–9-23
global timer registers, 9-23–9-28
interrupt source configuration registers, 9-24–9-26, 9-40–??
message registers, 9-34–??
non-accessible registers
interrupt pending register (IPR), 9-52
interrupt request register (IRR), 9-53
per-CPU registers, 9-46–9-51
performance monitor mask registers, 9-32–9-34, ??–9-34
summary registers, 9-28–??, 9-28–9-32, ??–9-32
PKEU
EU_GO, 10-96, 10-116, 10-143, 10-154, 10-160
interrupt control, 10-153
interrupt status, 10-151
key size, 10-147, 10-148
reset control, 10-149
status, 10-112, 10-150
PKEU data size, 10-149
processor version register (PVR), 23-29
RNG
data size, 10-156
interrupt control, 10-159
interrupt status, 10-158
mode, 10-156
reset control, 10-156
security engine (SEC)
KEU, 10-118–??
system version register (SVR), 23-30
trace buffer, 25-15–25-22
trigger out source register, 25-23
USB interface, 21-6–??
capability registers, 21-6–21-10
operational registers, 21-10
watchpoint monitor, 25-10–25-15
registers
SPI command,, 18-9
SPI event,, 18-6, 18-7
SPI mode,, 18-6, 18-12
Reset
core reset through PIC register, 9-22, 9-58
hard reset actions, 4-8
HRESET_REQ control, 23-30
operations, 4-8
power-on reset (POR)
configuration, see Power-on reset (POR), configuration
sequence of events, 4-9
requests from RapidIO and PCI Express, 23-28
signals summary, 4-2
see also Signals, reset
soft request, 23-27, 23-28
soft reset actions, 4-8
and reconfiguring the eTSEC, 14-161
RMON support, *see* eTSEC, modes of operation
RNG
data size register, 10-156
FIFO, 10-161
interrupt control register, 10-159
interrupt status register, 10-158
mode register, 10-156
reset control register, 10-156
RTC (real time clock) signal, 4-3, 4-26, 9-27, 9-28
RTS, see DUART_RTS[0:1]

## S

SCL (I$^2$C serial clock) signal, 11-3, 11-4
SD_RX[7:0]/$\overline{\text{SD\_RX}}$[7:0] (PCI Express serial data input and complement) signals, 17-5
SD_TX[7:0]/$\overline{\text{SD\_TX}}$[7:0] (PCI Express serial data output and complement) signals, 17-5
SDA (I$^2$C serial data) signal, 11-3, 11-4
Security engine (SEC)
register descriptions
KEU, 10-118–??
Serial data/clock, see I$^2$C interface, 11-1
Serial peripheral interface (SPI)
multi-master operation, 18-3
Signals
clock
RTC (real time clock), 4-3, 4-26, 9-27, 9-28
SYSCLK (system clock input), 4-3
complete signal listing
alphabetical reference, 3-9
configuration signals, sampled at POR, 3-15

see also Power-on reset (POR)
figure showing groupings, 3-1
output signal states at power-on reset, 3-17
reference by functional block, 3-2
DDR
MA[0:14] (address bus), 8-7
MBA[0:1] (logical bank address), 8-7
MCAS (column address strobe), 8-8
MCK[0:5] (DDR clock output complements), 8-10
MCK[0:5] (DDR clock outputs), 8-10
MCKE[0:3] (DDR clock enables), 8-10
MCS[0:3] (chip selects), 8-8
MDIC[0:1] (driver impedance calibration), 8-9
MDM[0:8] (SDRAM data output mask), 8-9
MDQS[0:8] (data bus strobes), 8-6, 8-61
MDVAL (debug mode data valid), 4-23, 25-3, 25-6
MECC[0:5] (error correcting code)
as debug signals, 25-3, 25-7
MECC[0:7] (error correcting code), 4-23, 8-7
MODT[0:3] (on-die termination), 8-9
MRAS (row address strobe), 8-8
MSRCID[0:4] (debug source ID), 4-23, 25-3, 25-7
MWE (write enable), 8-8, 8-9
DMA
DMA_DACK[0:3] (DMA acknowledge), 15-5
DMA_DDONE[0:3] (DMA done), 15-5
DMA_DREQ[0:3] (DMA request), 15-5
DUART
UART_CTS[0:1] (DUART clear to send), 12-1, 12-3
UART_RTS[0:1] (DUART request to send), 12-1, 12-3
UART_SIN [0:1] (DUART transmitter serial data in), 12-2, 12-3
UART_SOUT [0:1] (DUART transmitter serial data out), 12-2, 12-3
eTSEC
EC_GTX_CLK125 (eTSEC gigabit transmit 125 MHz source), 14-11
EC_MDC (eTSEC management data clock), 14-11
EC_MDIO (eTSEC management data input/output, BIDI), 14-11
FIFO interface signal summary, 14-159
TSEC*n*_COL (eTSEC 1–4 collision input), 14-10
TSEC*n*_CRS (eTSEC 1–4 carrier sense input/FIFO receiver flow control), 14-10
TSEC*n*_GTX_CLK (eTSEC 1–4 gigabit transmit clock), 14-10
TSEC*n*_RX_CLK (eTSEC 1–4 receive clock), 14-11
TSEC*n*_RX_DV (eTSEC 1–4 receive data valid), 14-11
TSEC*n*_RX_ER (eTSEC 1–4 receive error), 14-12
TSEC*n*_RXD[7:0] (eTSEC 1–4 receive data in), 14-12
TSEC*n*_TX_CLK (eTSEC 1–4 transmit clock in), 14-12
TSEC*n*_TX_EN (eTSEC 1–4 transmit data valid), 14-13
TSEC*n*_TX_ER (eTSEC 1–4 transmit error), 14-13
TSEC*n*_TXD[7:0] (eTSEC 1–4 transmit data out), 14-13
global utilities
ASLEEP, 23-2
CKSTP_IN (checkstop in), 23-2
CKSTP_OUT (checkstop out), 23-3
CLK_OUT, 23-3, 23-33
GPIO, 22-2
I$^2$C
SCL (serial clock), 11-3, 11-4
SDA (serial data), 11-3, 11-4
JTAG
TCK (JTAG test clock), 25-8
TDI (JTAG test data input), 25-8
TDO (JTAG test data output), 25-8
TMS (JTAG test mode select), 25-8
TRST (JTAG test reset), 25-9
LBC
LA[27:31] (non-multiplexed address), 13-7
LAD[0:31] (multiplexed address/data), 13-7
LALE (external address latch enable), 13-5, 13-43
LBCTL (data buffer control), 13-7, 13-46
LBS[0:3] (UPM byte select), 13-6
LCK[0:2] (clock), 13-8
LCS[0:7] (chip select), 13-6
LCS0 (LBC chip select 0), 13-58, 13-72
LDP[0:3] (data parity), 13-8
LGPL0 (GP line 0), 13-6
LGPL1 (GP line 1), 13-6
LGPL2 (GP line 2), 13-6
LGPL3 (GP line 3), 13-6
LGPL4 (GP line 4), 13-7
LGPL5 (GP line 5), 13-7
LGTA (GPCM transfer acknowledge), 13-7, 13-57
LOE (GPCM output enable), 13-6
LPBSE (parity byte select), 13-7
LSYNC_IN (DLL synchronization in), 13-8
LSYNC_OUT (DLL synchronization out), 13-8
LWE[0:3] (GPCM write enable), 13-6
MDVAL (debug mode data valid), 4-23, 13-8, 25-3, 25-6
MSRCID[0:4] (debug source ID), 4-23, 13-8, 25-3, 25-7
TA (data transfer acknowledge), 13-45
UPWAIT (UPM wait), 13-7, 13-75
other
TEST_SEL (factory test), 25-6
THERM[0:1] (thermal resistor access), 25-9
PCI Express
SD_RX[7:0]/SD_RX[7:0] (PCI Express serial data input and complement) signals, 17-5
SD_TX[7:0]/SD_TX[7:0] (PCI Express serial data output and complement) signals, 17-5
PCI/PCI-X

PCI_AD[63:0] (address/data bus), 16-6
PCI_C/BE[7:0] (command/byte enable), 16-7, 16-46, 16-48, 16-49, 16-65
PCI_DEVSEL (device select), 16-7, 16-48
PCI_FRAME (frame), 16-7, 16-45
PCI_GNT[4:0] (bus grant), 16-8, 16-42
PCI_IDSEL (initialization device), 16-8
PCI_IRDY (intitiator ready), 16-8, 16-45
PCI_PAR (parity), 16-9
PCI_PERR (parity error), 16-9, 16-66
PCI_REQ[4:0] (bus request), 16-9, 16-42
PCI_SERR (system error), 16-10, 16-66
PCI_STOP (stop), 16-10, 16-49
PCI_TRDY (target ready), 16-10, 16-45
PIC
IRQ[0:11], 9-8
IRQ_OUT, 9-8, 9-28
MCP, 9-8
reset
HRESET (hard reset), 4-2, 4-8
HRESET_REQ (hard reset request), 4-2, 11-18, 11-19
READY, 4-2, 25-23, 25-24
SRESET (soft reset), 4-2, 4-8
USB interface, 21-2–21-4
see also USB interface, signals
watchpoint monitor
TRIG_IN (watchpoint trigger in), 25-7, 25-11, 25-17
TRIG_OUT (watchpoint trigger out), 25-8, 25-23
Soft reset, 23-27, 23-28
SPCOM,, 18-9
SPI command register,, 18-7
SPI event register,, 18-6, 18-7
SPI,, 18-1
Split transaction isochronous transfer descriptor (siTD), see USB interface, split transaction isochronous transfer descriptor (siTD)
Split transactions, see USB interface, split transactions
SRAM, *see* L2 cache/SRAM, 6-27
SRESET (soft reset) signal, 4-2, 4-8
Stashing, *see* L2 cache/SRAM, stashing, 6-25
SYSCLK (system clock input) signal, 4-3
System interface, 21-40

## T

TA (LBC data transfer acknowledge) signal, 13-45
Target-disconnect, *see* PCI/PCI-X controller
TCK (JTAG test clock) signal, 25-8
TCR (timer control register), see e500 core, registers
TDI (JTAG test data input) signal, 25-8
TDO (JTAG test data output) signal, 25-8
Termination
PCI/PCI-X, termination of PCI transactions, 16-52
Test interface, see JTAG test access port
TEST_SEL (factory test) signal, 25-6
THERM[0:1] (thermal resistor access) signals, 25-9
Timing diagrams
PCI/PCI-X transactions, 16-50
TMS (JTAG test mode select) signal, 25-8
Trace buffer
and watchpoint monitor, block diagram, 25-1
as a second watchpoint monitor, 25-27
functional description, 25-27–25-29
initialization, 25-30
modes of triggering and arming, 25-4
overview, 25-1
register descriptions, 25-15–25-22
by acronym, see Register Index
see also Watchpoint monitor, 25-4
traced data formats relative to TBCR1[IFSEL]
DDR trace buffer entry, 25-28
ECM trace buffer entry, 25-27
PCI trace buffer entry, 25-28, 25-29
Transactions
PCI/PCI-X *see* PCI/PCI-X controller, transactions
TRIG_IN (watchpoint trigger in) signal, 25-7, 25-11, 25-17
TRIG_OUT (watchpoint trigger out) signal, 25-8, 25-23
TRST (JTAG test reset) signal, 25-9
TSEC*n*_COL (eTSEC 1–4 collision input) signals, 14-10
TSEC*n*_CRS (eTSEC 1–4 carrier sense input/FIFO receiver flow control) signals, 14-10
TSEC*n*_GTX_CLK (eTSEC 1–4 gigabit transmit clock) signals, 14-10
TSEC*n*_RX_CLK (eTSEC 1–4 receive clock) signals, 14-11
TSEC*n*_RX_DV (eTSEC 1–4 receive data valid) signals, 14-11
TSEC*n*_RX_ER (eTSEC 1–4 receive error) signals, 14-12
TSEC*n*_RXD[7:0] (eTSEC 1–4 receive data in) signals, 14-12
TSEC*n*_TX_CLK (eTSEC 1–4 transmit clock in) signals, 14-12
TSEC*n*_TX_EN (eTSEC 1–4 transmit data valid) signals, 14-13
TSEC*n*_TX_ER (eTSEC 1–4 transmit error) signals, 14-13
TSEC*n*_TXD[7:0] (eTSEC 1–4 transmit data out) signals, 14-13

## U

UART_CTS[0:1] (DUART clear to send) signals, 12-1, 12-3
UART_RTS[0:1] (DUART request to send) signals, 12-1, 12-3
UART_SIN [0:1] (DUART transmitter serial data in) signals, 12-2, 12-3
UART_SOUT [0:1] (DUART transmitter serial data out) signals, 12-2, 12-3

Universal asynchronous receiver/transmitter, see DUART
Universal serial bus, see USB interface
UPWAIT (LBC UPM wait) signal, 13-7, 13-75
USB interface
  asynchronous schedule, 21-41, 21-74
    adding queue heads, 21-75
    empty detection, 21-78
    list queue head pointer, 21-43
    organization, 21-43
    reclamation status bit, 21-79
    removing queue heads, 21-76
    traversal (start event), 21-79
  features, 21-2
  frame list
    link pointer format, 21-43
  functional descriptions
    DMA engine, 21-40
    FIFO RAM controller, 21-41
    PHY interface, 21-41
    system interface, 21-40
  host data structures, 21-41
  host operations, 21-61
    behavior during wake-up events, 21-65
    host controller initialization, 21-62
    periodic schedule, 21-69
    periodic schedule frame boundaries vs. bus frame boundaries, 21-67
    port power control (PPC), 21-63
    port suspend/resume, 21-64
    reporting over-current, 21-63
    schedule traversal rules, 21-66
    split transactions, 21-84
      execution state machine for isochronous, 21-104
      isochronous, 21-98
      periodic interrupt—Do-Complete-Split, 21-94
      periodic interrupt—Do-Start-Split, 21-93
      scheduling mechanisms for isochronous, 21-99
      see also USB interface, split transactions
      tests for execution, 21-94
      tracking progress for isochronous transfers, 21-102
    suspend/resume, 21-63
  isochronous (high-speed) transfer descriptor (iTD), 21-41, 21-44
    buffer page pointer list (plus), 21-46
    host controller operational model, 21-70
    managing transfers, 21-70
    next link pointer, 21-44
    periodic scheduling threshold, 21-73
    software operational model, 21-72
    transaction status and control list, 21-45
  memory map/register definitions, 21-4–21-5, ??–A-19
  modes of operation, 21-2
  overview, 21-2
  periodic frame list, 21-41, 21-42
  periodic frame span traversal node (FSTN), 21-60
    back path link pointer, 21-61
    normal path pointer, 21-61
  periodic schedule, 21-41, 21-42
    organization, 21-42
    rebalancing, 21-98
  queue element transfer descriptor (qTD), 21-41, 21-51
    alternate next pointer, 21-52
    buffer page pointer list, 21-56
    next pointer, 21-52
    token, 21-53
  queue heads, 21-41, 21-56
    endpoint capabilities/characteristics, 21-57
    horizontal link pointer, 21-57
    managing control/bulk/interrupt transfers, 21-79
      buffer pointer list use for data streaming with qTDs, 21-80
      ping control, 21-83
      to the periodic schedule, 21-82
      transfer complete interrupts, 21-82
    managing the QH[FrameTag] field, 21-97
    transfer overlay, 21-59
  registers
    by acronym, see Register Index
    capability registers, 21-6–21-10
    operational registers, 21-10
  signals, 21-2–21-4
    ULPI interface, 21-2
  split transaction isochronous transfer descriptor (siTD), 21-41, 21-47
    back link pointer, 21-51
    buffer pointer list (plus), 21-50
    endpoint capabilities/characteristics, 21-48
    next link pointer, 21-48
    transfer state, 21-49
  split transactions, 21-84
    asynchronous transfers, 21-84
      Do-Complete-Split state, 21-85
      Do-Start-Split state, 21-85
    interrupt, 21-86
      execution state machine, 21-92
      host controller operational model for FSTNs, 21-89
      scheduling mechanisms, 21-86
      software operational model for FSTNs, 21-91
      tracking progress for interrupt transfers, 21-92

## V

Voltage selection
  LBC signals, 23-31

## W

Watchpoint monitor
  and trace buffer, block diagram, 25-1
  functional description, 25-26
  initialization, 25-30
  modes of triggering and arming, 25-4
  overview, 25-1
  performance monitor events, 25-26
  register descriptions, 25-10–25-15
    by acronym, see Register Index
  second WM by using trace buffer, 25-27
  see also Trace buffer, 25-4
  signals summary, 25-5
    see also Signals, watchpoint, 25-5

## Z

ZBT SRAM interface (LBC), 13-99, 13-104