

---

# MPC8308 PowerQUICC II Pro Processor Reference Manual

MPC8308RM  
Rev. 0  
4/2010



## **How to Reach Us:**

### **Home Page:**

[www.freescale.com](http://www.freescale.com)

### **Web Support:**

<http://www.freescale.com/support>

### **USA/Europe or Locations Not Listed:**

Freescale Semiconductor, Inc.  
Technical Information Center, EL516  
2100 East Elliot Road  
Tempe, Arizona 85284  
+1-800-521-6274 or  
+1-480-768-2130  
[www.freescale.com/support](http://www.freescale.com/support)

### **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)  
[www.freescale.com/support](http://www.freescale.com/support)

### **Japan:**

Freescale Semiconductor Japan Ltd.  
Headquarters  
ARCO Tower 15F  
1-8-1, Shimo-Meguro, Meguro-ku  
Tokyo 153-0064  
Japan  
0120 191014 or  
+81 3 5437 9125  
[support.japan@freescale.com](mailto:support.japan@freescale.com)

### **Asia/Pacific:**

Freescale Semiconductor China Ltd.  
Exchange Building 23F  
No. 118 Jianguo Road  
Chaoyang District  
Beijing 100022  
China  
+86 010 5879 8000  
[support.asia@freescale.com](mailto:support.asia@freescale.com)

### **For Literature Requests Only:**

Freescale Semiconductor  
Literature Distribution Center  
+1-800 441-2447 or  
+1-303-675-2140  
Fax: +1-303-675-2150  
[LDCForFreescaleSemiconductor@hibbertgroup.com](mailto: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.

Freescale, the Freescale logo, CodeWarrior, ColdFire, PowerQUICC, StarCore, and Symphony are trademarks of Freescale Semiconductor, Inc. Reg. U.S. Pat. & Tm. Off. CoreNet, QorIQ, QUICC Engine, and VortiQa are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org. RapidIO is a registered trademark of the RapidIO Trade Association. IEEE 802.3, 802.3u, 802.3x, 802.3z, 802.3ac, and 802.11i are registered trademarks of the Institute of Electrical and Electronics Engineers, Inc. (IEEE). This product is not endorsed or approved by the IEEE.  
© 2010 Freescale Semiconductor, Inc.

Document Number: MPC8308RM  
Rev. 0, 4/2010



# Contents

Paragraph Number	Title	Page Number
<b>About This Book</b>		
	Audience .....	lvii
	Organization.....	lvii
	Suggested Reading.....	lix
	General Information.....	lix
	Related Documentation.....	lix
	Conventions .....	lx
	Signal Conventions .....	lx
	Acronyms and Abbreviations .....	lxi

## Chapter 1 Overview

1.1	MPC8308 Overview .....	1-1
1.2	MPC8308 Architecture Overview .....	1-7
1.2.1	e300 Core .....	1-7
1.2.2	DDR2 Memory Controller.....	1-10
1.2.3	Dual Enhanced Three-Speed Ethernet Controllers.....	1-10
1.2.4	SerDes PHY .....	1-11
1.2.5	PCI Express Interface .....	1-11
1.2.6	Universal Serial Bus (USB) 2.0.....	1-11
1.2.7	Enhanced Local Bus Controller (eLBC).....	1-12
1.2.8	Integrated Programmable Interrupt Controller (IPIC).....	1-14
1.2.9	I <sup>2</sup> C Interface.....	1-15
1.2.10	General Purpose DMA Controller .....	1-15
1.2.11	Dual Universal Asynchronous Receiver/Transmitter (DUART).....	1-16
1.2.12	Enhanced Secure Digital Host Controller (eSDHC).....	1-16
1.2.13	System Timers .....	1-17

## Chapter 2 Signal Descriptions

2.1	Signals Overview .....	2-1
2.2	Output Signal States During Reset .....	2-20

## Chapter 3 Memory Map

# Contents

Paragraph Number	Title	Page Number
3.1	Internal Memory-Mapped Registers .....	3-1
3.2	Accessing IMMR Memory from the Local Processor.....	3-1
3.3	IMMR Address Map .....	3-1

## Chapter 4 Reset, Clocking, and Initialization

4.1	External Signals .....	4-1
4.1.1	Reset Signals.....	4-1
4.1.2	Clock Signals .....	4-2
4.2	Functional Description.....	4-3
4.2.1	Reset Operations .....	4-3
4.2.2	Power-On Reset Flow .....	4-5
4.2.3	Hard Reset Flow .....	4-6
4.3	Reset Configuration .....	4-7
4.3.1	Reset Configuration Signals .....	4-7
4.3.2	Reset Configuration Words.....	4-9
4.3.3	Loading the Reset Configuration Words .....	4-16
4.4	Clocking .....	4-22
4.4.1	System Clock Domains.....	4-24
4.4.2	USB Clocking.....	4-25
4.4.3	Ethernet Clocking .....	4-25
4.5	Memory Map/Register Definitions .....	4-25
4.5.1	Reset Configuration Register Descriptions.....	4-25
4.5.2	Clock Configuration Registers.....	4-29

## Chapter 5 System Configuration

5.1	Local Memory Map Overview and Example .....	5-1
5.1.1	Address Translation and Mapping.....	5-3
5.1.2	Window into Configuration Space.....	5-3
5.1.3	Local Access Windows.....	5-4
5.1.4	Local Access Register Descriptions .....	5-5
5.1.5	Precedence of Local Access Windows .....	5-13
5.1.6	Configuring Local Access Windows .....	5-13
5.1.7	Distinguishing Local Access Windows from Other Mapping Functions .....	5-13
5.1.8	Outbound Address Translation and Mapping Windows.....	5-14
5.1.9	Inbound Address Translation and Mapping Windows .....	5-14
5.1.10	Internal Memory Map.....	5-14
5.1.11	Accessing Internal Memory from External Masters.....	5-15

# Contents

Paragraph Number	Title	Page Number
5.2	System Configuration .....	5-15
5.2.1	System Configuration Register Memory Map.....	5-15
5.2.2	System Configuration Registers .....	5-16
5.3	Software Watchdog Timer (WDT).....	5-31
5.3.1	WDT Overview.....	5-32
5.3.2	WDT Features.....	5-32
5.3.3	WDT Modes of Operation .....	5-32
5.3.4	WDT Memory Map/Register Definition .....	5-33
5.3.5	Functional Description.....	5-36
5.3.6	Initialization/Application Information (WDT Programming Guidelines).....	5-38
5.4	Real Time Clock (RTC) Module.....	5-38
5.4.1	Overview.....	5-38
5.4.2	Features.....	5-39
5.4.3	Assumptions.....	5-39
5.4.4	Modes of operation .....	5-39
5.4.5	External Signal Description .....	5-40
5.4.6	RTC Memory Map/Register Definition.....	5-40
5.4.7	Functional Description.....	5-44
5.4.8	RTC Reset Sequence.....	5-46
5.4.9	RTC Initialization Sequence .....	5-46
5.5	Periodic Interval Timer (PIT) .....	5-46
5.5.1	PIT Overview.....	5-46
5.5.2	PIT Features.....	5-47
5.5.3	PIT Modes of Operation .....	5-47
5.5.4	PIT External Signal Description .....	5-47
5.5.5	PIT Memory Map/Register Definition .....	5-48
5.5.6	Functional Description.....	5-51
5.5.7	PIT Programming Guidelines .....	5-52
5.6	General-Purpose Timers (GTM).....	5-52
5.6.1	GTM Overview .....	5-52
5.6.2	GTM Features .....	5-53
5.6.3	GTM Modes of Operation.....	5-54
5.6.4	GTM External Signal Description .....	5-55
5.6.5	GTM Memory Map/Register Definition.....	5-57
5.6.6	Functional Description.....	5-65
5.6.7	Initialization/Application Information (Programming Guidelines for GTM Registers)....	5-68
5.7	Power Management Control (PMC) .....	5-68
5.7.1	External Signal Description .....	5-69
5.7.2	PMC Memory Map/Register Definition .....	5-69
5.7.3	Functional Description.....	5-70

# Contents

Paragraph Number	Title	Page Number
<b>Chapter 6</b>		
<b>Arbiter and Bus Monitor</b>		
6.1	Overview .....	6-1
6.1.1	Coherent System Bus Overview .....	6-1
6.2	Arbiter Memory Map/Register Definition .....	6-2
6.2.1	Arbiter Configuration Register (ACR) .....	6-3
6.2.2	Arbiter Timers Register (ATR) .....	6-4
6.2.3	Arbiter Event Register (AER).....	6-5
6.2.4	Arbiter Interrupt Definition Register (AIDR).....	6-6
6.2.5	Arbiter Mask Register (AMR).....	6-7
6.2.6	Arbiter Event Attributes Register (AEATR).....	6-8
6.2.7	Arbiter Event Address Register (AEADR).....	6-9
6.2.8	Arbiter Event Response Register (AERR).....	6-10
6.3	Functional Description.....	6-11
6.3.1	Arbitration Policy .....	6-11
6.3.2	Bus Error Detection .....	6-14
6.4	Initialization/Applications Information .....	6-17
6.4.1	Initialization Sequence.....	6-17
6.4.2	Error Handling Sequence.....	6-17
<b>Chapter 7</b>		
<b>e300 Processor Core Overview</b>		
7.1	Overview .....	7-1
7.1.1	Features .....	7-3
7.1.2	Instruction Unit .....	7-6
7.1.3	Independent Execution Units.....	7-7
7.1.4	Completion Unit .....	7-8
7.1.5	Memory Subsystem Support.....	7-8
7.1.6	Bus Interface Unit (BIU) .....	7-10
7.1.7	System Support Functions .....	7-11
7.2	e300 Processor and System Version Numbers.....	7-13
7.3	PowerPC Architecture Implementation .....	7-13
7.4	Implementation-Specific Information.....	7-14
7.4.1	Register Model.....	7-14
7.4.2	Instruction Set and Addressing Modes .....	7-26
7.4.3	Cache Implementation .....	7-29
7.4.4	Interrupt Model .....	7-31
7.4.5	Memory Management.....	7-35
7.4.6	Instruction Timing .....	7-36

# Contents

Paragraph Number	Title	Page Number
7.4.7	Core Interface .....	7-37
7.4.8	Debug Features .....	7-39
7.5	Differences Between Cores.....	7-40

## Chapter 8 Integrated Programmable Interrupt Controller (IPIC)

8.1	Introduction.....	8-1
8.2	Features .....	8-4
8.3	Modes of Operation .....	8-4
8.3.1	Core Enable Mode .....	8-4
8.3.2	Core Disable Mode .....	8-4
8.4	External Signal Description .....	8-5
8.4.1	Overview.....	8-5
8.4.2	Detailed Signal Descriptions .....	8-5
8.5	Memory Map/Register Definition .....	8-5
8.5.1	System Global Interrupt Configuration Register (SICFR) .....	8-7
8.5.2	System Global Interrupt Vector Register (SIVCR).....	8-8
8.5.3	System Internal Interrupt Pending Registers (SIPNR_H and SIPNR_L).....	8-11
8.5.4	System Internal Interrupt Group A Priority Register (SIPRR_A).....	8-13
8.5.5	System Internal Interrupt Group B Priority Register (SIPRR_B) .....	8-14
8.5.6	System Internal Interrupt Group C Priority Register (SIPRR_C) .....	8-15
8.5.7	System Internal Interrupt Group D Priority Register (SIPRR_D).....	8-16
8.5.8	System Internal Interrupt Mask Register (SIMSR_H and SIMSR_L) .....	8-16
8.5.9	System Internal Interrupt Control Register (SICNR) .....	8-18
8.5.10	System External Interrupt Pending Register (SEPNR).....	8-20
8.5.11	System Mixed Interrupt Group A Priority Register (SMPRR_A).....	8-20
8.5.12	System Mixed Interrupt Group B Priority Register (SMPRR_B) .....	8-21
8.5.13	System External Interrupt Mask Register (SEMSR) .....	8-22
8.5.14	System External Interrupt Control Register (SECNR) .....	8-23
8.5.15	System Error Status Register (SERSR) .....	8-24
8.5.16	System Error Mask Register (SERMR).....	8-25
8.5.17	System Error Control Register (SERCR) .....	8-26
8.5.18	System External interrupt Polarity Control Register (SEPCCR) .....	8-26
8.5.19	System Internal Interrupt Force Registers (SIFCR_H and SIFCR_L) .....	8-27
8.5.20	System External Interrupt Force Register (SEFCR).....	8-28
8.5.21	System Error Force Register (SERFR) .....	8-29
8.5.22	System Critical Interrupt Vector Register (SCVCR) .....	8-29
8.5.23	System Management Interrupt Vector Register (SMVCR) .....	8-30
8.6	Functional Description.....	8-31
8.6.1	Interrupt Types .....	8-31

# Contents

Paragraph Number	Title	Page Number
8.6.2	Interrupt Configuration .....	8-32
8.6.3	Internal Interrupts Group Relative Priority .....	8-33
8.6.4	Mixed Interrupts Group Relative Priority .....	8-33
8.6.5	Highest Priority Interrupt .....	8-34
8.6.6	Interrupt Source Priorities .....	8-34
8.6.7	Masking Interrupt Sources .....	8-38
8.6.8	Interrupt Vector Generation and Calculation .....	8-39
8.6.9	Machine Check Interrupts .....	8-39
8.7	Message Shared Interrupts .....	8-40
8.7.1	Memory Map/Register Definition .....	8-40
8.7.2	Message Shared Registers .....	8-40

## Chapter 9 DDR Memory Controller

9.1	Introduction .....	9-1
9.2	Features .....	9-2
9.2.1	Modes of Operation .....	9-3
9.3	External Signal Descriptions .....	9-3
9.3.1	Signals Overview .....	9-3
9.3.2	Detailed Signal Descriptions .....	9-6
9.4	Memory Map/Register Definition .....	9-9
9.4.1	Register Descriptions .....	9-10
9.5	Functional Description .....	9-38
9.5.1	DDR SDRAM Interface Operation .....	9-41
9.5.2	DDR SDRAM Address Multiplexing .....	9-42
9.5.3	JEDEC Standard DDR SDRAM Interface Commands .....	9-44
9.5.4	DDR SDRAM Interface Timing .....	9-46
9.5.5	DDR SDRAM Mode-Set Command Timing .....	9-50
9.5.6	DDR SDRAM Registered DIMM Mode .....	9-50
9.5.7	DDR SDRAM Write Timing Adjustments .....	9-51
9.5.8	DDR SDRAM Refresh .....	9-52
9.5.9	DDR Data Beat Ordering .....	9-55
9.5.10	Page Mode and Logical Bank Retention .....	9-56
9.5.11	Error Checking and Correcting (ECC) .....	9-57
9.5.12	Error Management .....	9-59
9.6	Initialization/Application Information .....	9-59
9.6.1	DDR SDRAM Initialization Sequence .....	9-61

## Chapter 10 Enhanced Local Bus Controller



# Contents

Paragraph Number	Title	Page Number
10.1	Introduction.....	10-1
10.1.1	Overview.....	10-2
10.1.2	Features.....	10-2
10.1.3	Modes of Operation.....	10-3
10.2	External Signal Descriptions.....	10-4
10.3	Memory Map/Register Definition.....	10-7
10.3.1	Register Descriptions.....	10-8
10.4	Functional Description.....	10-39
10.4.1	Basic Architecture.....	10-40
10.4.2	General-Purpose Chip-Select Machine (GPCM).....	10-42
10.4.3	Flash Control Machine (FCM).....	10-53
10.4.4	User-Programmable Machines (UPMs).....	10-68
10.5	Initialization/Application Information.....	10-82
10.5.1	Interfacing to Peripherals in Different Address Modes.....	10-82
10.5.2	Interface to Different Port-Size Devices.....	10-83
10.5.3	Command Sequence Examples for NAND Flash EEPROM.....	10-84
10.5.4	Interfacing to Fast-Page Mode DRAM Using UPM.....	10-88
10.5.5	Interfacing to ZBT SRAM Using UPM.....	10-98

## Chapter 11 Enhanced Secure Digital Host Controller

11.1	Overview.....	11-1
11.2	Features.....	11-2
11.2.1	Data Transfer Modes.....	11-3
11.3	External Signal Description.....	11-3
11.4	Memory Map/Register Definition.....	11-4
11.4.1	DMA System Address Register (DSADDR).....	11-6
11.4.2	Block Attributes Register (BLKATTR).....	11-6
11.4.3	Command Argument Register (CMDARG).....	11-7
11.4.4	Transfer Type Register (XFERTYP).....	11-8
11.4.5	Command Response 0–3 (CMDRSP0–3).....	11-11
11.4.6	Buffer Data Port Register (DATPORT).....	11-13
11.4.7	Present State Register (PRSSTAT).....	11-13
11.4.8	Protocol Control Register (PROCTL).....	11-17
11.4.9	System Control Register (SYSCTL).....	11-20
11.4.10	Interrupt Status Register (IRQSTAT).....	11-23
11.4.11	Interrupt Status Enable Register (IRQSTATEN).....	11-27
11.4.12	Interrupt Signal Enable Register (IRQSIGEN).....	11-30
11.4.13	Auto CMD12 Error Status Register (AUTOC12ERR).....	11-32
11.4.14	Host Controller Capabilities (HOSTCAPBLT).....	11-34

# Contents

Paragraph Number	Title	Page Number
11.4.15	Watermark Level Register (WML).....	11-35
11.4.16	Force Event Register (FEVT).....	11-35
11.4.17	Host Controller Version Register (HOSTVER).....	11-37
11.4.18	DMA Control Register (DCR).....	11-37
11.5	Functional Description.....	11-37
11.5.1	Data Buffer .....	11-38
11.5.2	DMA CSB Interface .....	11-40
11.5.3	SD Protocol Unit.....	11-41
11.5.4	Clock & Reset Manager.....	11-43
11.5.5	Clock Generator.....	11-43
11.5.6	SDIO Card Interrupt .....	11-43
11.5.7	Card Insertion and Removal Detection.....	11-45
11.5.8	Power Management .....	11-45
11.6	Initialization/Application Information.....	11-46
11.6.1	Command Send and Response Receive Basic Operation.....	11-46
11.6.2	Card Identification Mode.....	11-47
11.6.3	Card Access .....	11-51
11.6.4	Switch Function .....	11-56
11.6.5	Commands for MMC/SD/SDIO .....	11-59
11.7	Software Restrictions.....	11-64
11.7.1	Initialization Active .....	11-64
11.7.2	Software Polling Procedure .....	11-64
11.7.3	Suspend Operation.....	11-64
11.7.4	Data Port Access.....	11-64
11.7.5	Multi-block Read .....	11-64

## Chapter 12 DMA Controller (DMAC)

12.1	Overview.....	12-1
12.1.1	Features.....	12-2
12.2	DMAC Memory Map/Register Definition .....	12-2
12.2.1	DMA Control Register (DMACR).....	12-3
12.3	DMA Error Status (DMAES) .....	12-6
12.3.1	DMA Enable Error Interrupt Register (DMAEEI).....	12-8
12.3.2	DMA Set Enable Error Interrupt (DMASEEI).....	12-9
12.3.3	DMA Clear Enable Error Interrupt (DMACEEI).....	12-9
12.3.4	DMA Clear Interrupt Request (DMACINT) .....	12-10
12.3.5	DMA Clear Error (DMACERR).....	12-11
12.3.6	DMA Set START Bit (DMASSRT).....	12-11
12.3.7	DMA Clear DONE Status (DMACDNE).....	12-12

# Contents

Paragraph Number	Title	Page Number
12.3.8	DMA Interrupt Request Register (DMAINT) .....	12-12
12.3.9	DMA Error Register (DMAERR).....	12-13
12.3.10	DMA General Purpose Output Register (DMAGPOR) .....	12-14
12.3.11	DMA Channel <i>n</i> Priority (DCHPRI <sub><i>n</i></sub> ), <i>n</i> = 0–15 .....	12-15
12.3.12	Transfer Control Descriptor (TCD) .....	12-16
12.4	Functional Description.....	12-24
12.4.1	DMA Microarchitecture .....	12-24
12.4.2	DMA Basic Data Flow .....	12-25
12.5	Initialization/Application Information.....	12-28
12.5.1	DMA Initialization.....	12-28
12.5.2	DMA Programming Errors .....	12-29
12.6	DMA Transfer.....	12-29
12.6.1	Single Request .....	12-29
12.6.2	Multiple Requests .....	12-30
12.7	TCD Status .....	12-32
12.7.1	Minor Loop Complete .....	12-32
12.7.2	Active Channel TCD Reads.....	12-32
12.7.3	Preemption status.....	12-32
12.8	Channel Linking .....	12-33
12.9	Programming during channel execution .....	12-33
12.9.1	Dynamic priority changing .....	12-33
12.9.2	Dynamic channel linking and dynamic scatter/gather .....	12-34

## Chapter 13 Universal Serial Bus Interface

13.1	Introduction.....	13-1
13.1.1	Overview.....	13-2
13.1.2	Features.....	13-2
13.1.3	Modes of Operation .....	13-2
13.2	External Signals .....	13-3
13.2.1	ULPI Interface .....	13-3
13.3	Memory Map/Register Definitions .....	13-4
13.3.1	Capability Registers.....	13-6
13.3.2	Operational Registers.....	13-10
13.4	Functional Description.....	13-44
13.4.1	System Interface .....	13-44
13.4.2	DMA Engine.....	13-45
13.4.3	FIFO RAM Controller .....	13-45
13.4.4	PHY Interface .....	13-45
13.5	Host Data Structures .....	13-46

# Contents

Paragraph Number	Title	Page Number
13.5.1	Periodic Frame List.....	13-46
13.5.2	Asynchronous List Queue Head Pointer.....	13-48
13.5.3	Isochronous (High-Speed) Transfer Descriptor (iTd).....	13-48
13.5.4	Split Transaction Isochronous Transfer Descriptor (siTD).....	13-52
13.5.5	Queue Element Transfer Descriptor (qTD) .....	13-56
13.5.6	Queue Head.....	13-62
13.5.7	Periodic Frame Span Traversal Node (FSTN).....	13-66
13.6	Host Operations .....	13-68
13.6.1	Host Controller Initialization .....	13-68
13.6.2	Power Port.....	13-69
13.6.3	Reporting Over-Current .....	13-69
13.6.4	Suspend/Resume .....	13-69
13.6.5	Schedule Traversal Rules.....	13-72
13.6.6	Periodic Schedule Frame Boundaries vs. Bus Frame Boundaries.....	13-73
13.6.7	Periodic Schedule .....	13-75
13.6.8	Managing Isochronous Transfers Using iTDs .....	13-76
13.6.9	Asynchronous Schedule.....	13-81
13.6.10	Managing Control/Bulk/Interrupt Transfers via Queue Heads.....	13-85
13.6.11	Ping Control .....	13-89
13.6.12	Split Transactions.....	13-90
13.6.13	Port Test Modes .....	13-118
13.6.14	Interrupts .....	13-119
13.7	Device Data Structures .....	13-123
13.7.1	Endpoint Queue Head.....	13-124
13.7.2	Endpoint Transfer Descriptor (dTD) .....	13-127
13.8	Device Operational Model.....	13-129
13.8.1	Device Controller Initialization .....	13-129
13.8.2	Port State and Control.....	13-130
13.8.3	Managing Endpoints .....	13-133
13.8.4	Managing Queue Heads.....	13-143
13.8.5	Managing Transfers with Transfer Descriptors .....	13-145
13.8.6	Servicing Interrupts.....	13-148
13.9	Deviations from the EHCI Specifications .....	13-149
13.9.1	Embedded Transaction Translator Function .....	13-150
13.9.2	Device Operation .....	13-153
13.9.3	Non-Zero Fields the Register File .....	13-154
13.9.4	SOF Interrupt .....	13-154
13.9.5	Embedded Design .....	13-154
13.9.6	Miscellaneous Variations from EHCI.....	13-154
13.10	Timing Diagrams .....	13-156

# Contents

Paragraph Number	Title	Page Number
<b>Chapter 14</b>		
<b>PCI Express Interface Controller</b>		
14.1	Introduction.....	14-1
14.1.1	MPC8308 as a PCI Express Initiator.....	14-3
14.1.2	MPC8308 as a PCI Express Target.....	14-3
14.1.3	Features.....	14-4
14.1.4	Modes of Operation.....	14-4
14.2	External Signal Descriptions.....	14-5
14.3	Memory Map/Register Definitions.....	14-5
14.3.1	PCI Express Memory Map.....	14-5
14.4	PCI Express Core Configuration Header Registers.....	14-14
14.4.1	Common PCI Express-Compatible Configuration Header Registers.....	14-14
14.4.2	Type 0 PCI Express-Compatible Configuration Header Registers.....	14-21
14.4.3	Type 1 PCI-Compatible Configuration Header Registers.....	14-27
14.4.4	PCI Express-Compatible Device-Specific Configuration Space Registers.....	14-36
14.4.5	PCI Express Extended Configuration Space.....	14-52
14.4.6	PCI Express Controller Internal Control and Status Registers (CSRs).....	14-62
14.4.7	PCI Express BAR Configuration Registers (EP Mode).....	14-72
14.4.8	PCI Express Extended Status and Control Registers.....	14-74
14.5	PCI Express CSB Bridge.....	14-76
14.5.1	PCI Express CSB Bridge Configuration Space.....	14-76
14.5.2	Global Registers.....	14-77
14.5.3	PCI Express Outbound PIO Registers.....	14-79
14.5.4	PCI Express Inbound PIO Registers.....	14-81
14.5.5	DMA Registers.....	14-83
14.5.6	Mailbox Registers.....	14-87
14.5.7	PCI Express Host Interrupt Registers.....	14-89
14.5.8	CSB System Interrupt Registers.....	14-94
14.5.9	PCI Express Power Management Registers.....	14-102
14.5.10	PCI Express Outbound Address Mapping Registers.....	14-103
14.5.11	PCI Express EP Inbound Address Translation Registers.....	14-106
14.5.12	PCI Express RC Inbound Address Mapping Registers.....	14-107
14.6	Functional Description.....	14-110
14.6.1	Architecture.....	14-111
14.6.2	Interrupts.....	14-121
14.6.3	Mailbox.....	14-123
14.6.4	Power Management.....	14-125
14.6.5	Hot Reset.....	14-126
14.7	Initialization/Application Information.....	14-126
14.7.1	Initialization Sequence.....	14-126

# Contents

Paragraph Number	Title	Page Number
14.8	DMA Functional Operation .....	14-127
14.8.1	DMA Descriptor Format.....	14-127
14.8.2	Write DMA .....	14-129
14.8.3	Read DMA .....	14-130
14.8.4	Descriptor-Based DMA .....	14-131

## Chapter 15 SerDes PHY

15.1	Introduction.....	15-1
15.1.1	Overview.....	15-1
15.1.2	Features.....	15-1
15.1.3	Mode of Operation.....	15-2
15.1.4	Clock.....	15-2
15.2	External Signals .....	15-2
15.3	Memory Map/Registers .....	15-3
15.3.1	SerDes Control Register 0 (SRDSCR0) .....	15-4
15.3.2	SerDes Control Register 1 (SRDSCR1) .....	15-6
15.3.3	SerDes Control Register 2 (SRDSCR2) .....	15-7
15.3.4	SerDes Control Register 3 (SRDSCR3) .....	15-8
15.3.5	SerDes Control Register 4 (SRDSCR4) .....	15-9
15.3.6	SerDes Reset Control Register (SRDSRSTCTL).....	15-10
15.4	Initialization Sequence and Reset .....	15-10
15.5	Power Management: Power Down .....	15-11

## Chapter 16 Enhanced Three-Speed Ethernet Controllers

16.1	Overview.....	16-1
16.2	Features .....	16-2
16.3	Modes of Operation .....	16-4
16.4	External Signals Description .....	16-5
16.4.1	Detailed Signal Descriptions .....	16-7
16.5	Memory Map/Register Definition .....	16-9
16.5.1	Top-Level Module Memory Map .....	16-10
16.5.2	Detailed Memory Map.....	16-10
16.5.3	Memory-Mapped Register Descriptions.....	16-21
16.6	Functional Description.....	16-120
16.6.1	Connecting to Physical Interfaces on Ethernet .....	16-121
16.6.2	Gigabit Ethernet Controller Channel Operation .....	16-124
16.6.3	TCP/IP Off-Load .....	16-139

# Contents

Paragraph Number	Title	Page Number
16.6.4	Quality of Service (QoS) Provision .....	16-144
16.6.5	Lossless Flow Control .....	16-154
16.6.6	Hardware Assist for IEEE Std. 1588 Compliant Timestamping .....	16-157
16.6.7	Buffer Descriptors.....	16-164
16.7	Initialization/Application Information.....	16-171
16.7.1	Interface Mode Configuration .....	16-172
16.7.2	MAC: Half-Duplex Collision on FCS of Short Frame .....	16-178

## Chapter 17 I<sup>2</sup>C Interface

17.1	Introduction.....	17-1
17.1.1	Features .....	17-2
17.1.2	Modes of Operation .....	17-2
17.2	External Signal Descriptions .....	17-3
17.2.1	Signal Overview .....	17-3
17.2.2	Detailed Signal Descriptions .....	17-3
17.3	Memory Map/Register Definition .....	17-4
17.3.1	Register Descriptions.....	17-5
17.4	Functional Description.....	17-10
17.4.1	Transaction Protocol .....	17-10
17.4.2	Arbitration Procedure .....	17-14
17.4.3	Handshaking .....	17-15
17.4.4	Clock Control.....	17-15
17.4.5	Boot Sequencer Mode.....	17-16
17.5	Initialization/Application Information.....	17-21
17.5.1	Interrupt Service Routine Flowchart.....	17-21
17.5.2	Initialization Sequence.....	17-23
17.5.3	Generation of START .....	17-23
17.5.4	Post-Transfer Software Response .....	17-23
17.5.5	Generation of STOP.....	17-24
17.5.6	Generation of Repeated START .....	17-24
17.5.7	Generation of SCL When SDA is Negated .....	17-24
17.5.8	Slave Mode Interrupt Service Routine.....	17-24

## Chapter 18 DUART

18.1	Overview.....	18-1
18.1.1	Features .....	18-2
18.1.2	Modes of Operation .....	18-2

# Contents

Paragraph Number	Title	Page Number
18.2	External Signal Descriptions .....	18-3
18.2.1	Signal Overview .....	18-3
18.2.2	Detailed Signal Descriptions .....	18-3
18.3	Memory Map/Register Definition .....	18-3
18.3.1	Register Descriptions.....	18-5
18.4	Functional Description.....	18-16
18.4.1	Serial Interface.....	18-17
18.4.2	Baud-Rate Generator Logic.....	18-18
18.4.3	Local Loopback Mode.....	18-19
18.4.4	Errors .....	18-19
18.4.5	FIFO Mode .....	18-19
18.5	DUART Initialization/Application Information .....	18-21

## Chapter 19 Serial Peripheral Interface

19.1	Overview.....	19-1
19.1.1	Features.....	19-2
19.1.2	SPI Transmission and Reception Process.....	19-2
19.1.3	Modes of Operation .....	19-3
19.2	External Signal Descriptions .....	19-6
19.2.1	Overview.....	19-6
19.2.2	Detailed Signal Descriptions .....	19-6
19.3	Memory Map/Register Definition .....	19-7
19.3.1	Register Descriptions.....	19-8
19.4	Initialization/Application Information.....	19-15
19.4.1	SPI Master Programming Example .....	19-15
19.4.2	SPI Slave Programming Example.....	19-15

## Chapter 20 JTAG/Testing Support

20.1	Overview.....	20-1
20.2	JTAG Signals .....	20-1
20.2.1	External Signal Descriptions .....	20-2
20.3	JTAG Registers and Scan Chains .....	20-3

## Chapter 21 General Purpose I/O (GPIO)

21.1	Introduction.....	21-1
------	-------------------	------



# Contents

Paragraph Number	Title	Page Number
21.1.1	Overview.....	21-1
21.1.2	Features.....	21-1
21.2	External Signal Description.....	21-2
21.2.1	Signals Overview.....	21-2
21.3	Memory Map/Register Definition.....	21-2
21.3.1	GPIO Direction Register (GPDIR).....	21-3
21.3.2	GPIO Open Drain Register (GPODR).....	21-3
21.3.3	GPIO Data Register (GPDAT).....	21-4
21.3.4	GPIO Interrupt Event Register (GPIER).....	21-4
21.3.5	GPIO Interrupt Mask Register (GPIMR).....	21-4
21.3.6	GPIO Interrupt Control Register (GPICR).....	21-5

## Appendix A

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

A.1	Local Access Windows.....	A-1
A.2	System Configuration Registers.....	A-2
A.3	Watchdog Timer (WDT).....	A-3
A.4	Real Time Clock (RTC).....	A-3
A.5	Periodic Interval Timer (PIT).....	A-3
A.6	General Purpose (Global) Timers (GTMs).....	A-4
A.7	Integrated Programmable Interrupt Controller (IPIC).....	A-5
A.8	System Arbiter.....	A-6
A.9	Reset Configuration.....	A-6
A.10	Clock Configuration.....	A-7
A.11	Power Management Controller (PMC).....	A-7
A.12	General Purpose I/O (GPIO).....	A-7
A.13	DDR Memory Controller.....	A-8
A.14	I <sup>2</sup> C Controller.....	A-9
A.15	DUART.....	A-9
A.16	Enhanced Local Bus Controller (eLBC).....	A-11
A.17	Serial Peripheral Interface (SPI).....	A-12
A.18	DMA Controller.....	A-13
A.19	PCI Express Controller.....	A-14
A.20	Enhanced Three-Speed Ethernet Controllers (eTSECs).....	A-21
A.21	SerDes PHY.....	A-31
A.22	Enhanced Secure Digital Host Controller (eSDHC).....	A-32
A.23	Universal Serial Bus (USB) Interface.....	A-32



# Contents

**Paragraph  
Number**

**Title**

**Page  
Number**

# Figures

Figure Number	Title	Page Number
1-1	MPC8308 Block Diagram.....	1-1
1-2	MPC8308 Integrated e300c3 Core Block Diagram .....	1-9
1-3	USB Controllers Port Configuration.....	1-12
2-1	MPC8308 Signal Groupings (1 of 2) .....	2-2
2-2	MPC8308 Signal Groupings (2 of 2) .....	2-3
4-1	Power-On Reset Flow .....	4-6
4-2	Hard Reset Flow.....	4-7
4-3	Reset Configuration Word Low Register (RCWLR) .....	4-10
4-4	Reset Configuration Word High Register (RCWHR) .....	4-12
4-5	EEPROM Data Format for Reset Configuration Words Preload Command .....	4-19
4-6	EEPROM Contents .....	4-20
4-7	Clock Subsystem Block Diagram .....	4-23
4-8	Reset Status Register (RSR).....	4-26
4-9	Reset Mode Register (RMR).....	4-27
4-10	Reset Protection Register (RPR).....	4-28
4-11	Reset Control Register (RCR).....	4-28
4-12	Reset Control Enable Register (RCER) .....	4-29
4-13	System PLL Mode Register .....	4-30
4-14	Output Clock Control Register (OCCR).....	4-31
4-15	System Clock Control Register (SCCR).....	4-32
5-1	Local Memory Map Example .....	5-2
5-2	Internal Memory Map Registers' Base Address Register (IMMRBAR).....	5-6
5-3	Alternate Configuration Base Address Register (ALTCBAR) .....	5-7
5-4	LBC Local Access Window <i>n</i> Base Address Registers (LBLAWBAR0–LBLAWBAR3) ....	5-7
5-5	LBC Local Access Window <i>n</i> Attributes Registers (LBLAWAR0–LBLAWAR3) .....	5-8
5-6	PCI Express 1 Local Access Window Base Address Register (PCIEXP1LAWBAR) .....	5-9
5-7	PCI Express 1 Local Access Window Attributes Register (PCIEXP1LAWAR) .....	5-10
5-8	DDR Local Access Window <i>n</i> Base Address Registers (DDRLAWBAR0–DDRLAWBAR1) ...	5-11
5-9	DDR Local Access Window <i>n</i> Attributes Registers (DDRLAWAR0–DDRLAWAR1).....	5-12
5-10	System General Purpose Register Low (SGPRL).....	5-16
5-11	System General Purpose Register High (SGPRH) .....	5-16
5-12	System Part and Revision ID Register (SPRIDR) .....	5-17
5-13	System Priority Configuration Register (SPCR) .....	5-18
5-14	System I/O Configuration Register Low (SICRL) .....	5-20
5-15	System I/O Configuration Register High (SICRH) .....	5-22
5-16	DDR Control Driver Register (DDRCDR).....	5-26
5-17	DDR Debug Status Register (DDRDSR).....	5-27

# Figures

Figure Number	Title	Page Number
5-18	PCI Express Controller Registers (PECR1).....	5-28
5-19	eSDHC Control Register (SDHCCR).....	5-30
5-20	RTC Control Register (RTCCR).....	5-31
5-21	Software Watchdog Timer High-Level Block Diagram .....	5-32
5-22	System Watchdog Control Register (SWCRR).....	5-33
5-23	System Watchdog Count Register (SWCNR).....	5-34
5-24	System Watchdog Service Register (SWSRR).....	5-35
5-25	Software Watchdog Timer Service State Diagram.....	5-36
5-26	Software Watchdog Timer Functional Block Diagram.....	5-37
5-27	RTC Block Diagram.....	5-39
5-28	Real Time Counter Control Register (RTCNR).....	5-41
5-29	Real Time Counter Load Register (RTLDR).....	5-42
5-30	Real Time Counter Prescale Register (RTPSR).....	5-42
5-31	Real Time Counter Register (RTCTR).....	5-43
5-32	Real Time Counter Event Register (RTEVR).....	5-43
5-33	Real Time Counter Alarm Register (RTALR).....	5-44
5-34	Real Time Clock Module Functional Block Diagram .....	5-45
5-35	Periodic Interval Timer High Level Block Diagram.....	5-47
5-36	Periodic Interval Timer Control Register (PTCNR).....	5-48
5-37	Periodic Interval Timer Load Register (PTLDR).....	5-49
5-38	Periodic Interval Timer Prescale Register (PTPSR).....	5-49
5-39	Periodic Interval Timer Counter Register (PTCTR).....	5-50
5-40	Periodic Interval Timer Event Register (PTEVR).....	5-50
5-41	Periodic Interval Timer Functional Block Diagram.....	5-51
5-42	Global Timers Block Diagram .....	5-53
5-43	Global Timers Configuration Register 1 (GTCFR1).....	5-58
5-44	Global Timers Configuration Register 2 (GTCFR2).....	5-59
5-45	Global Timers Mode Registers (GTMDR1–GTMDR4).....	5-61
5-46	Global Timers Reference Registers (GTRFR1–GTRFR4).....	5-62
5-47	Global Timers Capture Registers (GTCPR1–GTCPR4).....	5-62
5-48	Global Timers Counter Registers (GTCNR1–GTCNR4).....	5-63
5-49	Global Timers Event Registers (GTEVR1–GTEVR4).....	5-63
5-50	Global Timers Prescale Registers (GTPSR1–GTPSR4).....	5-64
5-51	Timers Non-Cascaded Mode Block Diagram .....	5-66
5-52	Timer Pair-Cascaded Mode Block Diagram .....	5-67
5-53	Timers Super-Cascaded Mode Block Diagram.....	5-67
5-54	Power Management Controller Configuration Register .....	5-69
6-1	Arbiter Configuration Register (ACR).....	6-3
6-2	Arbiter Timers Register (ATR).....	6-4
6-3	Arbiter Event Register (AER).....	6-5
6-4	Arbiter Interrupt Definition Register (AIDR).....	6-6

# Figures

Figure Number	Title	Page Number
6-5	Arbiter Mask Register (AMR) .....	6-7
6-6	Arbiter Event Attributes Register (AEATR) .....	6-8
6-7	Arbiter Event Address Register (AEADR) .....	6-10
6-8	Arbiter Event Response Register (AERR) .....	6-10
6-9	Address Bus Arbitration .....	6-11
6-10	An Example of Priority-Based Arbitration Algorithm .....	6-13
7-1	e300c3 Core Block Diagram .....	7-2
7-2	e300 Programming Model—Registers .....	7-16
7-3	e300c3 Data Cache Organization .....	7-30
7-4	Core Interface .....	7-38
8-1	Interrupt Sources Block Diagram .....	8-3
8-2	System Global Interrupt Configuration Register (SICFR) .....	8-7
8-3	System Global Interrupt Vector Register (SIVCR) .....	8-8
8-4	System Internal Interrupt Pending Register (SIPNR_H) .....	8-11
8-5	System Internal Interrupt Pending Register (SIPNR_L) .....	8-12
8-6	System Internal Interrupt Group A Priority Register (SIPRR_A) .....	8-14
8-7	System Internal Interrupt Group B Priority Register (SIPRR_B) .....	8-14
8-8	System Internal Interrupt Group C Priority Register (SIPRR_C) .....	8-15
8-9	System Internal Interrupt Group D Priority Register (SIPRR_D) .....	8-16
8-10	System Internal Interrupt Mask Register (SIMSR_H) .....	8-17
8-11	System Internal Interrupt Mask Register (SIMSR_L) .....	8-17
8-12	System Internal Interrupt Control Register (SICNR) .....	8-18
8-13	System External Interrupt Pending Register (SEPNR) .....	8-20
8-14	System Mixed Interrupt Group A Priority Register (SMPRR_A) .....	8-20
8-15	System Mixed Interrupt Group B Priority Register (SMPRR_B) .....	8-21
8-16	System External Interrupt Mask Register (SEMSR) .....	8-22
8-17	System External Interrupt Control Register (SECNR) .....	8-23
8-18	System Error Status Register (SERSR) .....	8-24
8-19	System Error Mask Register (SERMR) .....	8-25
8-20	System Error Control Register (SERCR) .....	8-26
8-21	System External Interrupt Polarity Control Register (SEPCR) .....	8-27
8-22	System Internal Interrupt Force Register (SIFCR_H) .....	8-27
8-23	System Internal Interrupt Force Register (SIFCR_L) .....	8-28
8-24	System External Interrupt Force Register (SEFCR) .....	8-28
8-25	System Error Status Register (SERFR) .....	8-29
8-26	System Critical Interrupt Vector Register (SCVCR) .....	8-30
8-27	System Management Interrupt Vector Register (SMVCR) .....	8-30
8-28	Interrupt Structure .....	8-32
8-29	DDR Interrupt Request Masking .....	8-39
8-30	Message Shared Interrupt Register (MSIRs) .....	8-41
8-31	Message Shared Interrupt Mask Register (MSIMR) .....	8-41

# Figures

Figure Number	Title	Page Number
8-32	Message Shared Interrupt Status Register (MSISR) .....	8-42
8-33	Message Shared Interrupt Index Register (MSIIR) .....	8-43
9-1	DDR Memory Controller Simplified Block Diagram.....	9-2
9-2	Chip Select Bounds Registers (CS <sub>n</sub> _BNDS).....	9-11
9-3	Chip Select Configuration Register (CS <sub>n</sub> _CONFIG).....	9-12
9-4	DDR SDRAM Timing Configuration 3 (TIMING_CFG_3) .....	9-13
9-5	DDR SDRAM Timing Configuration 0 (TIMING_CFG_0) .....	9-14
9-6	DDR SDRAM Timing Configuration 1 (TIMING_CFG_1) .....	9-16
9-7	DDR SDRAM Timing Configuration 2 Register (TIMING_CFG_2).....	9-18
9-8	DDR SDRAM Control Configuration Register (DDR_SDRAM_CFG).....	9-19
9-9	DDR SDRAM Control Configuration Register 2 (DDR_SDRAM_CFG_2).....	9-22
9-10	DDR SDRAM Mode Configuration Register (DDR_SDRAM_MODE).....	9-23
9-11	DDR SDRAM Mode 2 Configuration Register (DDR_SDRAM_MODE_2).....	9-24
9-12	DDR SDRAM Mode Control Register (DDR_SDRAM_MD_CNTL) .....	9-25
9-13	DDR SDRAM Interval Configuration Register (DDR_SDRAM_INTERVAL) .....	9-27
9-14	DDR SDRAM Data Initialization Configuration Register (DDR_DATA_INIT).....	9-28
9-15	DDR SDRAM Clock Control Configuration Register (DDR_SDRAM_CLK_CNTL).....	9-28
9-16	DDR Initialization Address Configuration Register (DDR_INIT_ADDR) .....	9-29
9-17	DDR IP Block Revision 1 (DDR_IP_REV1) .....	9-29
9-18	DDR IP Block Revision 2 (DDR_IP_REV2) .....	9-30
9-19	Memory Data Path Error Injection Mask High Register (DATA_ERR_INJECT_HI) .....	9-30
9-20	Memory Data Path Error Injection Mask Low Register (DATA_ERR_INJECT_LO).....	9-31
9-21	Memory Data Path Error Injection Mask ECC Register (ERR_INJECT).....	9-31
9-22	Memory Data Path Read Capture High Register (CAPTURE_DATA_HI).....	9-32
9-23	Memory Data Path Read Capture Low Register (CAPTURE_DATA_LO) .....	9-32
9-24	Memory Data Path Read Capture ECC Register (CAPTURE_ECC).....	9-33
9-25	Memory Error Detect Register (ERR_DETECT).....	9-33
9-26	Memory Error Disable Register (ERR_DISABLE).....	9-34
9-27	Memory Error Interrupt Enable Register (ERR_INT_EN).....	9-35
9-28	Memory Error Attributes Capture Register (CAPTURE_ATTRIBUTES).....	9-36
9-29	Memory Error Address Capture Register (CAPTURE_ADDRESS) .....	9-37
9-30	Single-Bit ECC Memory Error Management Register (ERR_SBE) .....	9-37
9-31	Typical Dual Data Rate SDRAM Internal Organization.....	9-39
9-32	Typical DDR SDRAM Interface Signals .....	9-39
9-33	Example 64-Mbyte DDR SDRAM Configuration With ECC .....	9-40
9-34	DDR SDRAM Burst Read Timing—ACTTORW = 3, MCAS Latency = 2 .....	9-48
9-35	DDR SDRAM Single-Beat (Double Word) Write Timing—ACTTOR .....	9-48
9-36	DDR SDRAM Single-Beat (Double Word) Write Timing—ACTTORW = 3.....	9-49
9-37	DDR SDRAM Clock Distribution Example for ¥8 DDR SDRAMs .....	9-49
9-38	DDR SDRAM Mode-Set Command Timing .....	9-50
9-39	Registered DDR SDRAM DIMM Burst Write Timing .....	9-51

# Figures

Figure Number	Title	Page Number
9-40	Write Timing Adjustments Example for Write Latency = 1 .....	9-52
9-41	DDR SDRAM Bank Staggered Auto Refresh Timing.....	9-53
9-42	DDR SDRAM Power-Down Mode .....	9-54
9-43	DDR SDRAM Self-Refresh Entry Timing .....	9-55
9-44	DDR SDRAM Self-Refresh Exit Timing .....	9-55
10-1	Enhanced Local Bus Controller Block Diagram.....	10-1
10-2	Base Registers (BR <sub>n</sub> ) .....	10-9
10-3	Option Registers (OR <sub>n</sub> ) in GPCM Mode.....	10-12
10-4	Option Registers (OR <sub>n</sub> ) in FCM Mode.....	10-14
10-5	Option Registers (OR <sub>n</sub> ) in UPM Mode .....	10-17
10-6	UPM Memory Address Register (MAR) .....	10-18
10-7	UPM Mode Registers (M <sub>x</sub> MR).....	10-19
10-8	Memory Refresh Timer Prescaler Register (MRTPR).....	10-21
10-9	UPM Data Register in UPM Mode (MDR) .....	10-22
10-10	FCM Data Register in FCM Mode (MDR).....	10-22
10-11	Special Operation Initiation Register (LSOR) .....	10-23
10-12	UPM Refresh Timer (LURT) .....	10-23
10-13	Transfer Error Status Register (LTESR) .....	10-24
10-14	Transfer Error Check Disable Register (LTEDR).....	10-26
10-15	Transfer Error Interrupt Enable Register (LTEIR).....	10-27
10-16	Transfer Error Attributes Register (LTEATR) .....	10-28
10-17	Transfer Error Address Register (LTEAR) .....	10-29
10-18	Transfer Error ECC Register (LTECCR) .....	10-29
10-19	Local Bus Configuration Register.....	10-30
10-20	Clock Ratio Register (LCRR) .....	10-31
10-21	Flash Mode Register .....	10-32
10-22	Flash Instruction Register .....	10-34
10-23	Flash Command Register .....	10-35
10-24	Flash Block Address Register .....	10-36
10-25	Flash Page Address Register, Small Page Device (OR <sub>x</sub> [PGS] = 0) .....	10-36
10-26	Flash Page Address Register, Large Page Device (OR <sub>x</sub> [PGS] = 1) .....	10-36
10-27	Flash Byte Count Register .....	10-38
10-28	Flash ECC Blockn Register (FECC0–FECC3).....	10-38
10-29	Basic Operation of Memory Controllers in the eLBC .....	10-40
10-30	Example of 8-Bit GPCM Writing 32 Bytes to Address 0x5420 (LCRR[PBYP] = 0).....	10-41
10-31	Basic eLBC Bus Cycle with TA, and LCS <sub>n</sub> .....	10-41
10-32	Enhanced Local Bus to GPCM Device Interface.....	10-42
10-33	GPCM Basic Read Timing (XACS = 0, ACS = 1x, TRLX = 0, CLKDIV = 4,8).....	10-43
10-34	GPCM General Read Timing Parameters .....	10-43
10-35	GPCM General Write Timing Parameters .....	10-45

# Figures

Figure Number	Title	Page Number
10-36	GPCM Basic Write Timing (XACS = 0, ACS = 00, CSNT = 1, SCY = 1, TRLX = 0, CLKDIV = 2, 4, 8).....	10-47
10-37	GPCM Relaxed Timing Back-to-Back Reads (XACS = 0, ACS = 1x, SCY = 1, CSNT = 0, TRLX = 1, EHTR = 0, CLKDIV = 4, 8)	10-49
10-38	GPCM Relaxed Timing Back-to-Back Writes (XACS = 0, ACS = 1x, SCY = 0, CSNT = 0, TRLX = 1, CLKDIV = 4, 8).....	10-49
10-39	GPCM Relaxed Timing Write (XACS = 0, ACS = 10, SCY = 0, CSNT = 1, TRLX = 1, CLKDIV = 4, 8).....	10-50
10-40	GPCM Relaxed Timing Write (XACS = 0, ACS = 00, SCY = 1, CSNT = 1, TRLX = 1, CLKDIV = 4, 8).....	10-50
10-41	GPCM Read Followed by Read (TRLX = 0, EHTR = 0, Fastest Timing).....	10-51
10-42	GPCM Read Followed by Write (TRLX = 0, EHTR = 1, One-Cycle Extended Hold Time on Reads) .....	10-51
10-43	External Termination of GPCM Access.....	10-52
10-44	Local Bus to 8-Bit FCM Device Interface .....	10-54
10-45	FCM Basic Page Read Timing (PGS = 1, CSCT = 0, CST = 0, CHT = 1, RST = 1, SCY = 0, TRLX = 0, EHTR = 1)	10-54
10-46	FCM Buffer RAM Memory Map for Small-Page (512-byte page) NAND Flash Devices	10-56
10-47	FCM Buffer RAM Memory Map for Large-Page (2-Kbyte page) NAND Flash Devices .	10-57
10-48	FCM ECC Calculation .....	10-57
10-49	ECC Placement in NAND Flash Spare Regions in Relation to FMR[ECCM] .....	10-58
10-50	FCM Instruction Sequencer Mechanism.....	10-59
10-51	Timing of FCM Command/Address and Write Data Cycles (for TRLX = 0, CHT = 0, CST = 1, SCY = 1, CLKDIV = 4*N).....	10-62
10-52	Example of FCM Command and Address Timing with Minimum Delay Parameters (for TRLX = 0, CHT = 0, CST = 0, SCY = 0, CLKDIV = 4*N).....	10-63
10-53	Example of FCM Command and Address Timing with Relaxed Parameters (for TRLX = 1, CHT = 0, CST = 1, SCY = 2, CLKDIV = 4*N).....	10-63
10-54	FCM Delay Prior to Sampling LFR $\bar{B}$ State .....	10-64
10-55	FCM Read Data Timing (for TRLX = 0, RST = 0, SCY = 1, CLKDIV = 4*N).....	10-64
10-56	FCM Read Data Timing with Extended Hold Time (for TRLX = 0, EHTR = 1, RST = 1, SCY = 1, CLKDIV = 4*N) .....	10-65
10-57	FCM Buffer RAM Memory Map During Boot Loading .....	10-67
10-58	User-Programmable Machine Functional Block Diagram.....	10-68
10-59	RAM Array Indexing .....	10-69
10-60	Memory Refresh Timer Request Block Diagram .....	10-70
10-61	UPM Clock Scheme for LCRR[CLKDIV] = 2.....	10-73
10-62	UPM Clock Scheme for LCRR[CLKDIV] = 4 or 8 .....	10-74
10-63	RAM Array and Signal Generation .....	10-74
10-64	RAM Word Fields .....	10-75



# Figures

Figure Number	Title	Page Number
10-65	$\overline{\text{LCSn}}$ Signal Selection .....	10-78
10-66	LBS Signal Selection .....	10-78
10-67	UPM Read Access Data Sampling.....	10-80
10-68	Effect of LUPWAIT Signal.....	10-81
10-69	GPCM Address Timings.....	10-82
10-70	GPCM Data Timings.....	10-83
10-71	Interface to Different Port-Size Devices .....	10-83
10-72	Single-Beat Read Access to FPM DRAM .....	10-89
10-73	Single-Beat Write Access to FPM DRAM .....	10-91
10-74	Burst Read Access to FPM DRAM Using LOOP (Two Beats).....	10-93
10-75	Refresh Cycle (CBR) to FPM DRAM .....	10-95
10-76	Exception Cycle .....	10-96
10-77	Interface to ZBT SRAM .....	10-98
11-1	System Connection of the eSDHC .....	11-1
11-2	eSDHC Block Diagram.....	11-2
11-3	DMA System Address Register (DSADDR) .....	11-6
11-4	Block Attributes Register (BLKATTR) .....	11-6
11-5	Command Argument Register (CMDARG) .....	11-7
11-6	Transfer Type Register (XFERTYP).....	11-8
11-7	Command Response 0–3 Register (CMDRSP <sub>n</sub> ) .....	11-11
11-8	Buffer Data Port Register (DATPORT) .....	11-13
11-9	Present State Register (PRSSTAT).....	11-13
11-10	Protocol Control Register (PROCTL).....	11-17
11-11	System Control Register (SYSCTL).....	11-20
11-12	Interrupt Status Register (IRQSTAT).....	11-24
11-13	Interrupt Status Enable Register (IRQSTATEN) .....	11-28
11-14	Interrupt Signal Enable Register (IRQSIGEN).....	11-30
11-15	Auto CMD12 Error Status Register (AUTOC12ERR).....	11-32
11-16	Host Capabilities Register (HOSTCAPBLT).....	11-34
11-17	Watermark Level Register (WML) .....	11-35
11-18	Force Event Register (FEVT) .....	11-36
11-19	Host Controller Version Register (HOSTVER) .....	11-37
11-20	eSDHC Buffer Scheme .....	11-38
11-21	Example of Dividing a Large Data Transfer .....	11-40
11-22	DMA CSB Interface Block.....	11-41
11-23	Command CRC Shift Register.....	11-42
11-24	Two Stages of Clock Divider .....	11-43
11-25	a) Card Interrupt Scheme; b) Card Interrupt Detection and Handling Procedure .....	11-45
11-26	Flow Diagram for Card Detection .....	11-47
11-27	Flow Chart for Reset of eSDHC and SD I/O Card .....	11-48
12-1	DMA Block Diagram.....	12-1

# Figures

Figure Number	Title	Page Number
12-2	DMA Control Register (DMACR) .....	12-4
12-3	DMA Error Status Register (DMAES) .....	12-7
12-4	DMA Enable Error Interrupt Register (DMAEEI) .....	12-8
12-5	DMA Set Enable Error Interrupt Register .....	12-9
12-6	DMA Clear Enable Error Interrupt Register.....	12-10
12-7	DMA Clear Interrupt Request Register .....	12-10
12-8	DMA Clear Error Register .....	12-11
12-9	DMA Set START Bit Register .....	12-11
12-10	DMA Clear DONE Status Register.....	12-12
12-11	DMA Interrupt Request Register Low (DMAINT) .....	12-13
12-12	DMA Error Register (DMAERR).....	12-14
12-13	DMA General Purpose Output Register (DMAGPOR).....	12-14
12-14	DMA Clear DONE Status Register.....	12-16
12-15	TCD Word 0 (TCDn.saddr) Field .....	12-17
12-16	TCD Word 1 (TCDn.{soff, smod, ssize, dmod, dsize}) Fields.....	12-18
12-17	TCD Word 2 (TCDn.nbytes) Field.....	12-19
12-18	TCD Word 3 (TCDn.slast) Field .....	12-19
12-19	TCD Word 4 (TCDn.daddr) Field.....	12-20
12-20	TCD Word 5 (TCDn.{citer, doff}) Fields .....	12-20
12-21	TCD Word 6 (TCDn.dlast_sga) Field .....	12-21
12-22	TCD Word 7 (TCDn.{biter, control/status}) Fields .....	12-22
12-23	DMA Operation—Part 1 .....	12-26
12-24	DMA Operation—Part 2.....	12-27
12-25	DMA Operation—Part 3.....	12-28
13-1	USB Interface Block Diagram .....	13-1
13-2	Capability Registers Length (CAPLENGTH).....	13-6
13-3	Host Controller Interface Version (HCIVERSION) .....	13-7
13-4	Host Controller Structural Parameters (HCSPARAMS).....	13-7
13-5	Host Control Capability Parameters (HCCPARAMS) .....	13-8
13-6	Device Interface Version (DCIVERSION) .....	13-9
13-7	Device Control Capability Parameters (DCCPARAMS).....	13-10
13-8	USB Command Register (USBCMD) .....	13-10
13-9	USB Status Register (USBSTS).....	13-13
13-10	USB Interrupt Enable (USBINTR) .....	13-15
13-11	USB Frame Index (FRINDEX).....	13-17
13-12	Periodic Frame List Base Address (PERIODICLISTBASE) .....	13-18
13-13	Device Address (DEVICEADDR).....	13-19
13-14	Current Asynchronous List Address (ASYNCLISTADDR) .....	13-19
13-15	Endpoint List Address (ENDPOINTLISTADDR).....	13-20
13-16	Master Interface Data Burst Size (BURSTSIZE) .....	13-21
13-17	Transmit FIFO Tuning Controls (TXFILLTUNING) .....	13-22

# Figures

Figure Number	Title	Page Number
13-18	ULPI Register Access (ULPI VIEWPORT) .....	13-23
13-19	Configure Flag Register (CONFIGFLAG) .....	13-24
13-20	Port Status and Control (PORTSC).....	13-25
13-21	OTG Status Control (OTGSC).....	13-30
13-22	USB Mode (USBMODE) .....	13-32
13-23	Endpoint Setup Status (ENDPTSETUPSTAT) .....	13-33
13-24	Endpoint Initialization (ENDPTPRIME) .....	13-34
13-25	Endpoint Flush (ENDPTFLUSH) .....	13-35
13-26	Endpoint Status (ENDPTSTATUS).....	13-35
13-27	Endpoint Complete (ENDPTCOMPLETE) .....	13-36
13-28	Endpoint Control 0 (ENDPTCTRL0) .....	13-37
13-29	Endpoint Control 1 to 5 (ENDPTCTRL <sub>n</sub> ) .....	13-38
13-30	Snoop 1 and Snoop 2 (SNOOP <sub>n</sub> ).....	13-40
13-31	Age Count Threshold (AGE_CNT_THRESH).....	13-41
13-32	Priority Control (PRI_CTRL) .....	13-42
13-33	System Interface Control Register (SI_CTRL).....	13-42
13-34	USB General-Purpose Register (CONTROL) .....	13-43
13-35	Periodic Schedule Organization .....	13-47
13-36	Frame List Link Pointer Format.....	13-47
13-37	Asynchronous Schedule Organization .....	13-48
13-38	Isochronous Transaction Descriptor (iT <sub>D</sub> ) .....	13-49
13-39	Split-Transaction Isochronous Transaction Descriptor (siTD) .....	13-52
13-40	Queue Element Transfer Descriptor (qTD).....	13-56
13-41	Queue Head Layout .....	13-62
13-42	Frame Span Traversal Node Structure .....	13-66
13-43	Derivation of Pointer into Frame List Array.....	13-72
13-44	General Format of Asynchronous Schedule List .....	13-73
13-45	Frame Boundary Relationship Between HS Bus and FS/LS Bus .....	13-73
13-46	Relationship of Periodic Schedule Frame Boundaries to Bus Frame Boundaries .....	13-74
13-47	Example Periodic Schedule .....	13-76
13-48	Example Association of iT <sub>D</sub> s to Client Request Buffer .....	13-79
13-49	Generic Queue Head Unlink Scenario .....	13-84
13-50	Asynchronous Schedule List with Annotation to Mark Head of List.....	13-85
13-51	Example Mapping of qTD Buffer Pointers to Buffer Pages .....	13-87
13-52	Host Controller Asynchronous Schedule Split-Transaction State Machine .....	13-90
13-53	Split Transaction, Interrupt Scheduling Boundary Conditions .....	13-93
13-54	General Structure of EHCI Periodic Schedule Utilizing Interrupt Spreading .....	13-94
13-55	Example Host Controller Traversal of Recovery Path via FSTNs.....	13-96
13-56	Split Transaction State Machine for Interrupt.....	13-99
13-57	Split Transaction, Isochronous Scheduling Boundary Conditions .....	13-106
13-58	siTD Scheduling Boundary Examples .....	13-108

# Figures

Figure Number	Title	Page Number
13-59	Split Transaction State Machine for Isochronous .....	13-111
13-60	Endpoint Queue Head Organization .....	13-124
13-61	Endpoint Queue Head Layout.....	13-125
13-62	Endpoint Transfer Descriptor (dTD).....	13-127
13-63	USB 2.0 Device States .....	13-131
13-64	Endpoint Queue Head Diagram .....	13-143
13-65	Software Link Pointers.....	13-145
13-66	ULPI Timing .....	13-156
13-67	Sending of RX CMD.....	13-157
13-68	ULPI Data Transmit (NOPID).....	13-157
13-69	ULPI Data Transmit (PID).....	13-158
13-70	ULPI Data Receive .....	13-158
13-71	ULPI Register Write.....	13-159
13-72	ULPI Register Read .....	13-159
14-1	PCI Express Controller Block Diagram.....	14-2
14-2	PCI Express PCI Express-Compatible Configuration Header Common Registers .....	14-14
14-3	PCI Express Vendor ID Register.....	14-15
14-4	PCI Express Device ID Register.....	14-15
14-5	PCI Express Command Register.....	14-16
14-6	PCI Express Status Register.....	14-17
14-7	PCI Express Revision ID Register .....	14-18
14-8	PCI Express Class Code Register .....	14-18
14-9	PCI Express Bus Cache Line Size Register .....	14-19
14-10	PCI Express Latency Timer Register .....	14-19
14-11	PCI Express Header Type Register .....	14-20
14-12	PCI Express PCI Express-Compatible Configuration Header—Type 0.....	14-21
14-13	32-Bit Base Address Registers (BAR0/BAR1) .....	14-22
14-14	64-Bit Low Memory Base Address Register (BAR2) .....	14-23
14-15	64-Bit High Memory Base Address Registers 3 and 5 (BAR3/BAR5).....	14-23
14-16	PCI Express Subsystem Vendor ID Register .....	14-24
14-17	PCI Express Subsystem ID Register .....	14-24
14-18	PCI Express Capabilities Pointer Register.....	14-25
14-19	PCI Express Interrupt Line Register .....	14-25
14-20	PCI Express Minimum Grant Register (MAX_GNT) .....	14-26
14-21	PCI Express Maximum Latency Register (MAX_LAT).....	14-26
14-22	PCI Express PCI Express-Compatible Configuration Header—Type 1 .....	14-27
14-23	PCI Express Primary Bus Number Register .....	14-27
14-24	PCI Express Secondary Bus Number Register .....	14-28
14-25	PCI Express Subordinate Bus Number Register.....	14-28
14-26	PCI Express I/O Base Register .....	14-29
14-27	PCI Express I/O Limit Register .....	14-29

# Figures

Figure Number	Title	Page Number
14-28	PCI Express Secondary Status Register .....	14-30
14-29	PCI Express Memory Base Register .....	14-30
14-30	PCI Express Memory Limit Register .....	14-31
14-31	PCI Express Prefetchable Memory Base Register .....	14-31
14-32	PCI Express Prefetchable Memory Limit Register .....	14-32
14-33	PCI Express Prefetchable Base Upper 32-Bit Register .....	14-32
14-34	PCI Express Prefetchable Limit Upper 32-Bit Register .....	14-33
14-35	PCI Express I/O Base Upper 16-Bit Register .....	14-33
14-36	PCI Express I/O Limit Upper 16-Bit Register .....	14-34
14-37	PCI Express Capabilities Pointer Register .....	14-34
14-38	PCI Express Interrupt Line Register .....	14-35
14-39	PCI Express Bridge Control Register .....	14-35
14-40	PCI Express-Compatible Device-Specific Configuration Space .....	14-36
14-41	PCI Express Power Management Capability ID Register .....	14-37
14-42	PCI Express Power Management Next Capabilities Pointer .....	14-37
14-43	PCI Express Power Management Capabilities Register .....	14-38
14-44	PCI Express Power Management Status and Control Register .....	14-38
14-45	PCI Express Power Management Data Register .....	14-39
14-46	PCI Express Capability ID Register .....	14-39
14-47	PCI Express Next Capabilities Pointer .....	14-40
14-48	PCI Express Capabilities Register .....	14-40
14-49	PCI Express Device Capabilities Register .....	14-41
14-50	PCI Express Device Control Register .....	14-42
14-51	PCI Express Device Status Register .....	14-43
14-52	PCI Express Link Capabilities Register .....	14-43
14-53	PCI Express Link Control Register .....	14-44
14-54	PCI Express Link Status Register .....	14-45
14-55	PCI Express Slot Capabilities Register .....	14-45
14-56	PCI Express Slot Control Register .....	14-46
14-57	PCI Express Slot Status Register .....	14-47
14-58	PCI Express Root Control Register .....	14-47
14-59	PCI Express Root Status Register .....	14-48
14-60	PCI Express Capability ID Register .....	14-49
14-61	PCI Express MSI Message Control Register .....	14-49
14-62	PCI Express MSI Message Address Register .....	14-50
14-63	PCI Express MSI Message Upper Address Register .....	14-50
14-64	PCI Express MSI Message Data Register .....	14-50
14-65	PCI Express Extended Configuration Space .....	14-52
14-66	PCI Express Advanced Error Reporting Capability ID Register .....	14-53
14-67	PCI Express Uncorrectable Error Status Register .....	14-53
14-68	PCI Express Uncorrectable Error Mask Register .....	14-54

## Figures

Figure Number	Title	Page Number
14-69	PCI Express Uncorrectable Error Severity Register .....	14-55
14-70	PCI Express Correctable Error Status Register.....	14-56
14-71	PCI Express Correctable Error Mask Register .....	14-57
14-72	PCI Express Advanced Error Capabilities and Control Register.....	14-58
14-73	PCI Express Header Log Register .....	14-59
14-74	PCI Express Root Error Command Register.....	14-60
14-75	PCI Express Root Error Status Register.....	14-60
14-76	PCI Express Error Source Identification Register .....	14-61
14-77	PCI Express LTSSM State Status Register (PEX_LTSSM_STAT) .....	14-62
14-78	PCI Express N_FTS Control Register .....	14-64
14-79	PCI Express ACK Replay Timeout Register .....	14-64
14-80	PCI Express Core Clock Ratio Register (PEX_GCLK_RATIO).....	14-66
14-81	PCI Express Power Management Timer Register (PEX_PM_TIMER) .....	14-66
14-82	PCI Express PME Time-Out Register (PEX_PME_TIMEOUT) .....	14-67
14-83	PCI Express ASPM Request Timer Register .....	14-67
14-84	PCI Express Subsystem Vendor ID Update Register (PEX_SSVID_UPDATE).....	14-68
14-85	PCI Express Device Capabilities Update Register .....	14-69
14-86	PCI Express Link Capabilities Update Register .....	14-70
14-87	PCI Express Slot Capabilities Update Register .....	14-71
14-88	PCI Express Configuration Ready Register (PEX_CFG_READY) .....	14-72
14-89	PCI Express BAR Size Low Configuration Register (PEX_BAR_SIZEL) .....	14-72
14-90	PCI Express BAR Select Configuration Register (PEX_BAR_SEL) .....	14-73
14-91	PCI Express BAR Prefetch Configuration Register (PEX_BAR_PF).....	14-74
14-92	PCI Express PME_To_Ack Timeout Register (PEX_PME_TO_ACK_TOR).....	14-74
14-93	PME_To_Ack Status Register (PEX_PME_TO_ACK_SR).....	14-75
14-94	PCI Express PCI Interrupt Mask Register (PEX_SS_INTR_MASK).....	14-76
14-95	PCI Express CSB Bridge Control Register (PEX_CSB_CTRL).....	14-77
14-96	PCI Express DMA Descriptor Timer Register (PEX_DMA_DSTMR) .....	14-78
14-97	PCI Express CSB Bridge Status Register (PEX_CSB_STAT) .....	14-79
14-98	PCI Express Outbound PIO Control Register (PEX_CSB_OBCTRL) .....	14-80
14-99	PCI Express Outbound PIO Status Register (PEX_CSB_OBSTAT).....	14-81
14-100	PCI Express Inbound PIO Control Register (PEX_CSB_IBCTRL) .....	14-82
14-101	PCI Express Inbound PIO Status Register (PEX_CSB_IBSTAT) .....	14-82
14-102	PCI Express Write DMA Control Register (PEX_WDMA_CTRL) .....	14-83
14-103	PCI Express Write DMA First Address Register (PEX_WDMA_ADDR) .....	14-84
14-104	PCI Express Write DMA Status Register (PEX_WDMA_STAT).....	14-84
14-105	PCI Express Read DMA Control Register (PEX_RDMA_CTRL) .....	14-85
14-106	PCI Express Read DMA First Address Register (PEX_RDMA_ADDR).....	14-86
14-107	PCI Express Read DMA Status Register (PEX_RDMA_STAT).....	14-86
14-108	PCI Express Outbound Mailbox Control Register (PEX_OMBCR) .....	14-87
14-109	MPCI Express Outbound Mailbox Data Register (PEX_OMBDR).....	14-88

# Figures

Figure Number	Title	Page Number
14-110	PCI Express Inbound Mailbox Control Register (PEX_IMBCR) .....	14-88
14-111	PCI Express Inbound Mailbox Data Register (PEX_IMBDR).....	14-89
14-112	PCI Express Host Interrupt Enable Register (PEX_HIER) .....	14-89
14-113	PCI Express Host Interrupt Status Register (PEX_HISR).....	14-90
14-114	PCI Express Host Outbound PIO Interrupt Vector Register (PEX_HOPIVR).....	14-91
14-115	PCI Express Host Inbound PIO Interrupt Vector Register (PEX_HIPIVR) .....	14-92
14-116	PCI Express Host Write DMA Interrupt Vector Register (PEX_HWDIVR).....	14-92
14-117	PCI Express Host Read DMA Interrupt Vector Register (PEX_HRDIVR) .....	14-93
14-118	PCI Express Host Miscellaneous Interrupt Vector Register (PEX_HMIVR).....	14-93
14-119	CSB System PIO Interrupt Enable Register (PEX_CSPIER).....	14-94
14-120	CSB System Write DMA Interrupt Enable Register (PEX_CSWDIER) .....	14-95
14-121	CSB System Read DMA Interrupt Enable Register (PEX_CSRDIER) .....	14-95
14-122	CSB System Miscellaneous Interrupt Enable Register (PEX_CSMIER).....	14-96
14-123	CSB System PIO Interrupt Status Register (PEX_CSPISR) .....	14-98
14-124	CSB System Write DMA Interrupt Status Register (PEX_CSWDISR) .....	14-99
14-125	CSB System Read DMA Interrupt Status Register (PEX_CSRDISR).....	14-99
14-126	CSB System Miscellaneous Interrupt Status Register (PEX_CSMISR) .....	14-100
14-127	PCI Express PM Control Register (PEX_PM_CTRL) .....	14-102
14-128	PCI Express Outbound Window Attributes Register <i>n</i> (PEX_OWAR0–PEX_OWAR3).	14-103
14-129	PCI Express Outbound Window Base Address Register <i>n</i> (PEX_OWBAR0–PEX_OWBAR3) 14-104	
14-130	PCI Express Outbound Window Translation Address Register Low <i>n</i> (PEX_OWTARL0–PEX_OWTARL3).....	14-105
14-131	PCI Express Outbound Window Translation Address Register High <i>n</i> (PEX_OWTARH0–PEX_OWTARH3).....	14-106
14-132	PCI Express EP Inbound Window Translation Address Register <i>n</i> (PEX_EPIWTAR0–PEX_EPIWTAR3) .....	14-107
14-133	PCI Express RC Inbound Window Attributes Register <i>n</i> (PEX_RCIWAR0–PEX_RCIWAR3) . 14-108	
14-134	PCI Express RC Inbound Window Translation Address Register <i>n</i> (PEX_RCIWTAR0–PEX_RCIWTAR3).....	14-109
14-135	PCI Express RC Inbound Window Base Address Register Low <i>n</i> (PEX_RCIWBARL0–PEX_RCIWBARL3).....	14-109
14-136	CI Express RC Inbound Window Base Address Register High <i>n</i> (PEX_RCIWBARH0–PEX_RCIWBARH3) .....	14-110
14-137	Requestor/Completer Relationship .....	14-110
14-138	PCI Express High-Level Layering .....	14-111
14-139	PCI Express Packet Flow .....	14-111
14-140	Outbound Byte Swapping .....	14-114
14-141	Example—How to Generate WAKE#.....	14-126
14-142	DMA Descriptor Format.....	14-128

# Figures

Figure Number	Title	Page Number
14-143	<i>n</i> -Way Chain Descriptor Organization in Host Memory .....	14-131
14-144	Block Descriptor Organization in Host Memory .....	14-132
15-1	SerDes PHY Block Diagram.....	15-1
15-2	SerDes Control Register 0 (SRDSCR0).....	15-4
15-3	SerDes <sub><i>n</i></sub> Control Register 1 (SRDS <sub><i>n</i></sub> CR1).....	15-6
15-4	SerDes Control Register 2 (SRDSCR2).....	15-7
15-5	SerDes Control Register 3 (SRDSCR3).....	15-8
15-6	SerDes Control Register 4 (SRDSCR4).....	15-9
15-7	SerDes Reset Control Register (SRDSRSTCTL) .....	15-10
16-1	eTSEC Block Diagram.....	16-2
16-2	TSEC_ID Register .....	16-21
16-3	TSEC_ID2 Register .....	16-22
16-4	IEVENT Register Definition .....	16-23
16-5	IMASK Register Definition .....	16-27
16-6	EDIS Register Definition .....	16-28
16-7	ECNTRL Register Definition .....	16-30
16-8	PTV Register Definition.....	16-32
16-9	DMACTRL Register.....	16-32
16-10	TBIPA Register Definition.....	16-34
16-11	TCTRL Register Definition .....	16-34
16-12	TSTAT Register Definition .....	16-36
16-13	DFVLAN Register Definition.....	16-40
16-14	TXIC Register Definition.....	16-41
16-15	TQUEUE Register Definition.....	16-42
16-16	TR03WT Register Definition.....	16-43
16-17	TR47WT Register Definition.....	16-43
16-18	TBPTR0–TBPTR7 Register Definition .....	16-44
16-19	TBASE Register Definition .....	16-45
16-20	TMR_TXTS <sub><i>n</i></sub> _ID Register Definition .....	16-45
16-21	TMR_TXTS <sub><i>n</i></sub> _H/L Register Definition.....	16-46
16-22	RCTRL Register Definition .....	16-46
16-23	RSTAT Register Definition.....	16-49
16-24	RXIC Register Definition .....	16-50
16-25	RQUEUE Register Definition.....	16-51
16-26	RBIFX Register Definition .....	16-52
16-27	Receive Queue Filer Table Address Register Definition .....	16-54
16-28	Receive Queue Filer Table Control Register Definition .....	16-54
16-29	Receive Queue Filer Table Property IDs 0, 2–15 Register Definition.....	16-55
16-30	Receive Queue Filer Table Property ID1 Register Definition .....	16-56
16-31	MRBLR Register Definition .....	16-59
16-32	RBPTR0–RBPTR7 Register Definition .....	16-60



# Figures

Figure Number	Title	Page Number
16-33	RBASE Register Definition .....	16-60
16-34	TMR_RXTS_H/L Register Definition.....	16-61
16-35	MACCFG1 Register Definition .....	16-64
16-36	MACCFG2 Register Definition .....	16-66
16-37	IPGIFG Register Definition .....	16-68
16-38	Half-Duplex Register Definition.....	16-69
16-39	Maximum Frame Length Register Definition.....	16-70
16-40	MII Management Configuration Register Definition .....	16-70
16-41	MIIMCOM Register Definition .....	16-71
16-42	MIIMADD Register Definition .....	16-72
16-43	MII Mgmt Control Register Definition.....	16-72
16-44	MIIMSTAT Register Definition .....	16-73
16-45	MII Mgmt Indicator Register Definition .....	16-73
16-46	Interface Status Register Definition .....	16-74
16-47	MAC Station Address Part 1 Register Definition .....	16-74
16-48	MAC Station Address Part 2 Register Definition .....	16-75
16-49	MAC Exact Match Address <i>n</i> Part 1 Register Definition .....	16-76
16-50	MAC Exact Match Address <i>x</i> Part 2 Register Definition .....	16-76
16-51	Transmit and Receive 64-Byte Frame Register Definition.....	16-78
16-52	Transmit and Receive 65- to 127-Byte Frame Register Definition .....	16-78
16-53	Transmit and Received 128- to 255-Byte Frame Register Definition .....	16-79
16-54	Transmit and Received 256- to 511-Byte Frame Register Definition.....	16-79
16-55	Transmit and Received 512- to 1023-Byte Frame Register Definition .....	16-80
16-56	Transmit and Received 1024- to 1518-Byte Frame Register Definition .....	16-80
16-57	Transmit and Received 1519- to 1522-Byte VLAN Frame Register Definition .....	16-81
16-58	Receive Byte Counter Register Definition.....	16-81
16-59	Receive Packet Counter Register Definition .....	16-81
16-60	Receive FCS Error Counter Register Definition.....	16-82
16-61	Receive Multicast Packet Counter Register Definition .....	16-82
16-62	Receive Broadcast Packet Counter Register Definition .....	16-83
16-63	Receive Control Frame Packet Counter Register Definition .....	16-83
16-64	Receive Pause Frame Packet Counter Register Definition .....	16-84
16-65	Receive Unknown OPCode Packet Counter Register Definition .....	16-84
16-66	Receive Alignment Error Counter Register Definition.....	16-85
16-67	Receive Frame Length Error Counter Register Definition .....	16-85
16-68	Receive Code Error Counter Register Definition .....	16-86
16-69	Receive Carrier Sense Error Counter Register Definition .....	16-86
16-70	Receive Undersize Packet Counter Register Definition .....	16-87
16-71	Receive Oversize Packet Counter Register Definition .....	16-87
16-72	Receive Fragments Counter Register Definition .....	16-88
16-73	Receive Jabber Counter Register Definition.....	16-88

# Figures

Figure Number	Title	Page Number
16-74	Receive Dropped Packet Counter Register Definition .....	16-89
16-75	Transmit Byte Counter Register Definition .....	16-89
16-76	Transmit Packet Counter Register Definition .....	16-90
16-77	Transmit Multicast Packet Counter Register Definition .....	16-90
16-78	Transmit Broadcast Packet Counter Register Definition .....	16-91
16-79	Transmit Pause Control Frame Counter Register Definition .....	16-91
16-80	Transmit Deferral Packet Counter Register Definition.....	16-92
16-81	Transmit Excessive Deferral Packet Counter Register Definition.....	16-92
16-82	Transmit Single Collision Packet Counter Register Definition .....	16-93
16-83	Transmit Multiple Collision Packet Counter Register Definition.....	16-93
16-84	Transmit Late Collision Packet Counter Register Definition .....	16-94
16-85	Transmit Excessive Collision Packet Counter Register Definition .....	16-94
16-86	Transmit Total Collision Counter Register Definition .....	16-95
16-87	Transmit Drop Frame Counter Register Definition .....	16-95
16-88	Transmit Jabber Frame Counter Register Definition .....	16-96
16-89	Transmit FCS Error Counter Register Definition .....	16-96
16-90	Transmit Control Frame Counter Register Definition .....	16-97
16-91	Transmit Oversized Frame Counter Register Definition .....	16-97
16-92	Transmit Undersize Frame Counter Register Definition .....	16-98
16-93	Transmit Fragment Counter Register Definition .....	16-98
16-94	Carry Register 1 (CAR1) Register Definition.....	16-99
16-95	Carry Register 2 (CAR2) Register Definition.....	16-100
16-96	Carry Mask Register 1 (CAM1) Register Definition.....	16-101
16-97	Carry Mask Register 2 (CAM2) Register Definition.....	16-103
16-98	Receive Filer Rejected Packet Counter Register Definition .....	16-104
16-99	IGADDR $n$ Register Definition .....	16-105
16-100	GADDR $n$ Register Definition.....	16-105
16-101	ATTR Register Definition .....	16-106
16-102	ATTRELI Register Definition.....	16-107
16-103	RQPRM Register Definition .....	16-108
16-104	RFBPTR0–RFBPTR7 Register Definition.....	16-108
16-105	TMR_CTRL Register Definition .....	16-109
16-106	TMR_TEVENT Register Definition.....	16-112
16-107	TMR_PEVENT Register Definition .....	16-114
16-108	TMR_PEMASK Register Definition .....	16-114
16-109	TMR_CNT_H Register Definition .....	16-116
16-110	TMR_ADD Register Definition.....	16-116
16-111	TMR_ACC Register Definition .....	16-117
16-112	TMR_PRSC Register Definition .....	16-117
16-113	TMROFF_H/L Register Definition .....	16-118
16-114	TMR_ALARM1-2_H/L Register Definition .....	16-118

# Figures

Figure Number	Title	Page Number
16-115	TMR_FIPER $n$ Register Definition .....	16-120
16-116	TMR_ETTS1-2_H/L Register Definition .....	16-120
16-117	eTSEC-MII Connection .....	16-121
16-118	eTSEC-RGMII Connection.....	16-122
16-119	Definition of Custom Preamble Sequence .....	16-129
16-120	Definition of Received Preamble Sequence.....	16-129
16-121	Ethernet Address Recognition Flowchart .....	16-131
16-122	Location of Frame Control Blocks for TOE Parameters .....	16-140
16-123	Transmit Frame Control Block .....	16-141
16-124	Receive Frame Control Block.....	16-142
16-125	Structure of the Receive Queue Filer Table .....	16-147
16-126	1588 Timer Design Partition .....	16-158
16-127	Ethernet Sampling Points for 1588 .....	16-158
16-128	PTP Packet Format.....	16-160
16-129	Buffer Format for Transmit Time-Stamp Insertion.....	16-162
16-130	Transmit Frame Control Block .....	16-162
16-131	Example of eTSEC Memory Structure for BDs .....	16-165
16-132	Buffer Descriptor Ring.....	16-165
16-133	Transmit Buffer Descriptor .....	16-166
16-134	Receive Buffer Descriptor.....	16-169
16-135	Mapping of RxBDs to a C Data Structure .....	16-170
17-1	I <sup>2</sup> C Block Diagram.....	17-1
17-2	I <sup>2</sup> C Address Register (I2CADR).....	17-5
17-3	I <sup>2</sup> C Frequency Divider Register (I2CFDR) .....	17-5
17-4	I <sup>2</sup> C Control Register (I2CCR).....	17-6
17-5	I <sup>2</sup> C Status Register (I2CSR) .....	17-8
17-6	I <sup>2</sup> C Data Register (I2CDR).....	17-9
17-7	I <sup>2</sup> C Digital Filter Sampling Rate Register (I2CDFSRR).....	17-10
17-8	I <sup>2</sup> C Interface Transaction Protocol.....	17-11
17-9	EEPROM Contents .....	17-19
17-10	EEPROM Data Format for One Register Preload Command.....	17-20
17-11	Example I <sup>2</sup> C Interrupt Service Routine Flowchart .....	17-22
18-1	UART Block Diagram .....	18-1
18-2	Receiver Buffer Registers (URBR1 and URBR2) .....	18-5
18-3	Transmitter Holding Registers (UTHR1 and UTHR2) .....	18-6
18-4	Divisor Most Significant Byte Registers (UDMB1 and UDMB2) .....	18-6
18-5	Divisor Least Significant Byte Registers (UDLB1 and UDLB2).....	18-7
18-6	Interrupt Enable Registers (UIER1 and UIER2).....	18-8
18-7	Interrupt ID Registers (UIIR1 and UIIR2).....	18-9
18-8	FIFO Control Registers (UFCR1 and UFCR2).....	18-10
18-9	Line Control Register (ULCR1 and ULCR2) .....	18-11

# Figures

Figure Number	Title	Page Number
18-10	Modem Control Register (UMCR1 and UMCR2).....	18-12
18-11	Line Status Register (ULSR1 and ULSR2) .....	18-13
18-12	Scratch Register (USCR) .....	18-14
18-13	Alternate Function Register (UAFR).....	18-14
18-14	DMA Status Register (UDSR) .....	18-15
18-15	UART Bus Interface Transaction Protocol Example .....	18-17
19-1	SPI Block Diagram .....	19-1
19-2	Single-Master/Multi-Slave Configuration .....	19-3
19-3	Multiple-Master Configuration .....	19-5
19-4	SPMODE-SPI Mode Register Definition .....	19-8
19-5	SPI Transfer Format with SPMODE[CP] = 0.....	19-10
19-6	SPI Transfer Format with SPMODE[CP] = 1 .....	19-10
19-7	SPIE—SPI Event Register Definition.....	19-11
19-8	SPIM—SPI Mask Register Definition .....	19-12
19-9	SPI Command Register Definition .....	19-13
19-10	SPI Transmit Data Hold Register Definition .....	19-13
19-11	SPI Receive Data Hold Register Definition.....	19-14
19-12	Example SPMODE[REV] = 0 SPMODE[LEN] = 7 LSB Sent First.....	19-14
19-13	Example SPMODE[REV] = 1 SPMODE[LEN] = 7 MSB Sent First.....	19-14
19-14	Example SPMODE[REV] = 1 SPMODE[LEN] = 15 MSB Sent First.....	19-14
19-15	Example SPMODE[REV] = 0 SPMODE[LEN] = 15 LSB Sent First.....	19-15
20-1	JTAG Interface Block Diagram .....	20-1
21-1	GPIO Module Block Diagram .....	21-1
21-2	GPIO Direction Register (GPDIR) .....	21-3
21-3	GPIO Open Drain Register (GPODR) .....	21-3
21-4	GPIO Data Register (GPDAT) .....	21-4
21-5	GPIO Interrupt Event Register (GPIER) .....	21-4
21-6	GPIO Interrupt Mask Register (GPIMR).....	21-5
21-7	GPIO Interrupt Control Register (GPICR) .....	21-5

# Tables

Table Number	Title	Page Number
i	Acronyms and Abbreviated Terms.....	lxi
2-1	MPC8308 Signal Reference by Functional Block .....	2-3
2-2	MPC8308 Signal Reference by Alphabetical Order .....	2-12
2-3	Output Signal States During System Reset .....	2-20
3-1	IMMR Memory Map .....	3-2
4-1	System Control Signals .....	4-1
4-2	External Clock Signals .....	4-2
4-3	Reset Causes .....	4-3
4-4	Reset Actions .....	4-4
4-5	Reset Configuration Words Source .....	4-8
4-6	Selecting Reset Configuration Input Signals .....	4-9
4-7	RCWLR Bit Settings .....	4-10
4-8	System PLL VCO Division.....	4-11
4-9	System PLL Ratio .....	4-11
4-10	Reset Configuration Word High Bit Settings .....	4-12
4-11	Boot Memory Space.....	4-13
4-12	Boot Sequencer Configuration.....	4-14
4-13	Boot ROM Location.....	4-15
4-14	eTSEC1 Mode Configuration .....	4-15
4-15	eTSEC2 Mode Configuration .....	4-16
4-16	e300 Core True Little-Endian .....	4-16
4-17	Local Bus Configuration EEPROM Addresses .....	4-17
4-18	Local Bus Reset Configuration Words Data Structure.....	4-17
4-19	Local Bus Controller Setting When Loading RCW.....	4-18
4-20	RCW Values Corresponding to Hard Coded Options .....	4-21
4-21	Hard Coded Reset Configuration Word Low Fields Values .....	4-21
4-22	Hard-Coded Reset Configuration Word High Field Values .....	4-22
4-23	Configurable Clock Units .....	4-24
4-24	Reset Configuration and Status Registers Memory Map.....	4-25
4-25	Reset Status Register Field Descriptions .....	4-26
4-26	RMR Field Descriptions .....	4-27
4-27	RPR Bit Descriptions .....	4-28
4-28	RCR Bit Settings .....	4-29
4-29	RCER Bit Settings .....	4-29
4-30	Clock Configuration Registers Memory Map.....	4-29
4-31	System PLL Mode Register Bit Settings .....	4-30
4-32	OCCR Bit Settings .....	4-31
4-33	SCCR Bit Descriptions .....	4-32

# Tables

Table Number	Title	Page Number
5-1	Local Access Windows Target Interface .....	5-1
5-2	Local Access Windows Example .....	5-2
5-3	Format of Window Definitions .....	5-3
5-4	Local Access Register Memory Map .....	5-4
5-5	IMMRBAR Bit Settings .....	5-6
5-6	ALTCBAR Bit Settings .....	5-7
5-7	LBLAWBAR0–LBLAWBAR3 Bit Settings .....	5-8
5-8	LBLAWBAR0[BASE_ADDR] Reset Value .....	5-8
5-9	LBLAWAR0–LBLAWAR3 Bit Settings .....	5-8
5-10	LBLAWAR0[EN] Reset Value .....	5-9
5-11	PCIEXP1LAWBAR Bit Settings .....	5-10
5-12	PCIEXP1LAWAR Bit Settings .....	5-10
5-13	DDRLAWBAR0–DDRLAWBAR1 Bit Settings .....	5-11
5-14	DDRLAWBAR0[BASE_ADDR] Reset Value .....	5-11
5-15	DDRLAWAR0–DDRLAWAR1 Bit Settings .....	5-12
5-16	DDRLAWAR0[EN] Reset Value .....	5-13
5-17	Overlapping Local Access Windows .....	5-13
5-18	System Configuration Register Memory Map .....	5-15
5-19	SGPRL Bit Settings .....	5-16
5-20	SGPRH Bit Settings .....	5-16
5-21	SPRIDR Bit Settings .....	5-17
5-22	PARTID Coding .....	5-17
5-23	REVID Coding .....	5-17
5-24	SPCR Bit Settings .....	5-18
5-25	SICRL Bit Settings .....	5-20
5-26	SICRH Bit Settings .....	5-22
5-27	SICRH[27–31] Bit Settings .....	5-25
5-28	DDRCDR Field Descriptions .....	5-27
5-29	DDRDSR Field Descriptions .....	5-28
5-30	PECR Field Description .....	5-29
5-31	SDHCCR Field Description .....	5-30
5-32	RTCCR Field Description .....	5-31
5-33	WDT Register Address Map .....	5-33
5-34	SWCRR Bit Settings .....	5-34
5-35	SWCNR Bit Settings .....	5-35
5-36	SWSRR Bit Settings .....	5-35
5-37	RTC External Signals .....	5-40
5-38	RTC Register Address Map .....	5-40
5-39	RTCNR Bit Settings .....	5-41
5-40	RTLDR Bit Settings .....	5-42
5-41	RTPSR Bit Settings .....	5-42

## Tables

Table Number	Title	Page Number
5-42	RTCTR Bit Settings .....	5-43
5-43	RTEVR Bit Settings .....	5-43
5-44	RTALR Bit Settings .....	5-44
5-45	PIT Signal Properties .....	5-47
5-46	PIT External Signal—Detailed Signal Descriptions .....	5-47
5-47	PIT Register Address Map .....	5-48
5-48	PTCNR Bit Settings .....	5-48
5-49	PTLDR Bit Settings .....	5-49
5-50	PTPSR Bit Settings .....	5-50
5-51	PTCTR Bit Settings .....	5-50
5-52	PTEVR Bit Settings .....	5-51
5-53	GTM Signal Properties .....	5-55
5-54	GTM External Signals—Detailed Signal Descriptions .....	5-56
5-55	GTM Register Address Map .....	5-57
5-56	GTCFR1 Bit Settings .....	5-58
5-57	GTCFR2 Bit Settings .....	5-60
5-58	GTMDR Bit Settings .....	5-61
5-59	GTRFR Bit Settings .....	5-62
5-60	GTCPR <sub>n</sub> Bit Settings .....	5-63
5-61	GTCNR Bit Settings .....	5-63
5-62	GTEVR <sub>n</sub> Bit Settings .....	5-64
5-63	GTPSR <sub>n</sub> Bit Settings .....	5-64
5-64	System Control Signals—Detailed Signal Descriptions .....	5-69
5-65	Power Management Controller Registers Memory Map .....	5-69
5-66	PMCCR Bit Settings .....	5-70
5-67	Software-Controller Power-Down States—Basic Description .....	5-70
6-1	Arbiter Register Map .....	6-2
6-2	ACR Field Descriptions .....	6-3
6-3	ATR Field Descriptions .....	6-5
6-4	AER Field Descriptions .....	6-5
6-5	AIDR Field Descriptions .....	6-6
6-6	AMR Field Descriptions .....	6-7
6-7	AEATR Field Descriptions .....	6-8
6-8	AEADR Field Descriptions .....	6-10
6-9	AERR Field Descriptions .....	6-10
6-10	Address Only Transaction Type Encoding .....	6-15
6-11	Reserved Transaction Type Encoding .....	6-16
6-12	Illegal Transaction Type Encoding .....	6-16
7-1	Device Revision Level Cross-Reference .....	7-13
7-2	MSR Bit Descriptions .....	7-18
7-3	e300 HID0 Bit Descriptions .....	7-22

## Tables

Table Number	Title	Page Number
7-4	Using HID0[ECLK] and HID0[SBCLK] to Configure <i>clk_out</i> .....	7-24
7-5	HID1 Bit Descriptions .....	7-25
7-6	e300HID2 Bit Descriptions.....	7-25
7-7	Interrupt Classifications .....	7-33
7-8	Exceptions and Interrupts.....	7-33
7-9	Differences Between e300 and G2_LE Cores .....	7-40
8-1	IPIC Signal Properties.....	8-5
8-2	IPIC External Signals—Detailed Signal Descriptions.....	8-5
8-3	IPIC Register Address Map .....	8-6
8-4	SICFR Field Descriptions .....	8-7
8-5	SIVCR Field Descriptions .....	8-9
8-6	IVEC/CVEC/MVEC Field Definition .....	8-9
8-7	SIPNR_H/SIFCR_H/SIMSR_H Bit Assignments.....	8-11
8-8	SIPNR_H Field Descriptions .....	8-12
8-9	SIPNR_L/SIFCR_L/SIMSR_L Bit Assignments .....	8-12
8-10	SIPNR_L Field Descriptions .....	8-13
8-11	SIPRR_A Field Descriptions .....	8-14
8-12	SIPRR_B Field Descriptions .....	8-15
8-13	SIPRR_C Field Descriptions .....	8-15
8-14	SIPRR_D Field Descriptions .....	8-16
8-15	SIMSR_H Field Descriptions .....	8-17
8-16	SIMSR_L Field Descriptions.....	8-18
8-17	SICNR Field Descriptions .....	8-18
8-18	SEPNR Field Descriptions.....	8-20
8-19	SMPRR_A Field Descriptions .....	8-21
8-20	SMPRR_B Field Descriptions .....	8-21
8-21	SEMSR Field Descriptions .....	8-22
8-22	SECNR Field Descriptions .....	8-23
8-23	SERSR/SERM/SERFR Bit Assignments.....	8-24
8-24	SERSR Field Descriptions .....	8-25
8-25	SERM Field Descriptions.....	8-25
8-26	SERCR Field Descriptions.....	8-26
8-27	SEPCR Field Descriptions .....	8-27
8-28	SIFCR_H Field Descriptions .....	8-27
8-29	SIFCR_L Field Descriptions.....	8-28
8-30	SEFCR Field Descriptions .....	8-29
8-31	SERFR Field Descriptions .....	8-29
8-32	SCVCR Field Descriptions .....	8-30
8-33	SMVCR Field Descriptions .....	8-31
8-34	Interrupt Source Priority Levels.....	8-34
8-35	Message Shared Registers Address Map .....	8-40



# Tables

Table Number	Title	Page Number
8-36	MSIRs Field Descriptions .....	8-41
8-37	MSIMR Field Descriptions .....	8-41
8-38	MSISR Field Descriptions .....	8-42
8-39	MSIIR Field Descriptions .....	8-43
9-1	DDR Memory Interface Signal Summary .....	9-4
9-2	Memory Address Signal Mappings.....	9-5
9-3	Memory Interface Signals—Detailed Signal Descriptions .....	9-6
9-4	Clock Signals—Detailed Signal Descriptions .....	9-9
9-5	DDR Memory Controller Memory Map.....	9-9
9-6	CS <sub>n</sub> _BNDS Field Descriptions.....	9-11
9-7	CS <sub>n</sub> _CONFIG Field Descriptions .....	9-12
9-8	TIMING_CFG_3 Field Descriptions .....	9-13
9-9	TIMING_CFG_0 Field Descriptions .....	9-14
9-10	TIMING_CFG_1 Field Descriptions .....	9-16
9-11	TIMING_CFG_2 Field Descriptions .....	9-18
9-12	DDR_SDRAM_CFG Field Descriptions .....	9-20
9-13	DDR_SDRAM_CFG_2 Field Descriptions .....	9-22
9-14	DDR_SDRAM_MODE Field Descriptions .....	9-24
9-15	DDR_SDRAM_MODE_2 Field Descriptions .....	9-24
9-16	DDR_SDRAM_MD_CNTL Field Descriptions.....	9-25
9-17	Settings of DDR_SDRAM_MD_CNTL Fields .....	9-26
9-18	DDR_SDRAM_INTERVAL Field Descriptions .....	9-27
9-19	DDR_DATA_INIT Field Descriptions .....	9-28
9-20	DDR_SDRAM_CLK_CNTL Field Descriptions .....	9-28
9-21	DDR_INIT_ADDR Field Descriptions .....	9-29
9-22	DDR_IP_REV1 Field Descriptions .....	9-29
9-23	DDR_IP_REV2 Field Descriptions .....	9-30
9-24	DATA_ERR_INJECT_HI Field Descriptions.....	9-30
9-25	DATA_ERR_INJECT_LO Field Descriptions .....	9-31
9-26	ERR_INJECT Field Descriptions .....	9-31
9-27	CAPTURE_DATA_HI Field Descriptions.....	9-32
9-28	CAPTURE_DATA_LO Field Descriptions.....	9-32
9-29	CAPTURE_ECC Field Descriptions .....	9-33
9-30	ERR_DETECT Field Descriptions .....	9-33
9-31	ERR_DISABLE Field Descriptions.....	9-34
9-32	ERR_INT_EN Field Descriptions .....	9-35
9-33	CAPTURE_ATTRIBUTES Field Descriptions .....	9-36
9-34	CAPTURE_ADDRESS Field Descriptions .....	9-37
9-35	ERR_SBE Field Descriptions .....	9-37
9-36	Byte Lane to Data Relationship .....	9-41
9-37	Supported DDR2 SDRAM Device Configurations .....	9-42

# Tables

Table Number	Title	Page Number
9-38	DDR2 Address Multiplexing for 32-Bit Data Bus with Interleaving and Partial Array Self Refresh Disabled .....	9-42
9-39	DDR2 Address Multiplexing for 16-Bit Data Bus.....	9-43
9-40	Example of Address Multiplexing for 32-Bit Data Bus Interleaving between Two Banks with Partial Array Self Refresh Disabled.....	9-44
9-41	DDR SDRAM Command Table.....	9-45
9-42	DDR SDRAM Interface Timing Intervals .....	9-46
9-43	DDR SDRAM Power-Saving Modes Refresh Configuration.....	9-54
9-44	Memory Controller–Data Beat Ordering .....	9-56
9-45	DDR SDRAM ECC Syndrome Encoding .....	9-57
9-46	DDR SDRAM ECC Syndrome Encoding (Check Bits) .....	9-58
9-47	Memory Controller Errors .....	9-59
9-48	Memory Interface Configuration Register Initialization Parameters.....	9-60
10-1	Signal Properties—Summary .....	10-4
10-2	Enhanced Local Bus Controller Detailed Signal Descriptions .....	10-5
10-3	Enhanced Local Bus Controller Registers .....	10-7
10-4	BR <sub>n</sub> Field Descriptions.....	10-9
10-5	Reset value of OR0 Register.....	10-10
10-6	Memory Bank Sizes in Relation to Address Mask .....	10-11
10-7	OR <sub>n</sub> —GPCM Field Descriptions .....	10-12
10-8	OR <sub>n</sub> —FCM Field Descriptions .....	10-14
10-9	OR <sub>n</sub> —UPM Field Descriptions .....	10-17
10-10	MAR Field Descriptions .....	10-18
10-11	M <sub>x</sub> MR Field Descriptions.....	10-19
10-12	MRTPR Field Descriptions.....	10-21
10-13	MDR Field Description.....	10-22
10-14	LSOR Field Description.....	10-23
10-15	LURT Field Descriptions .....	10-24
10-16	LTESR Field Descriptions .....	10-25
10-17	LTEDR Field Descriptions.....	10-26
10-18	LTEIR Field Descriptions .....	10-27
10-19	LTEATR Field Descriptions.....	10-28
10-20	LTEAR Field Descriptions.....	10-29
10-21	LTECCR Field Descriptions .....	10-29
10-22	LBCR Field Descriptions.....	10-30
10-23	LCRR Field Descriptions.....	10-32
10-24	FMR Field Descriptions.....	10-33
10-25	FIR Field Descriptions .....	10-35
10-26	FCR Field Descriptions.....	10-35
10-27	FBAR Field Descriptions.....	10-36
10-28	FPAR Field Descriptions, Small Page Device (OR <sub>x</sub> [PGS] = 0).....	10-37

# Tables

Table Number	Title	Page Number
10-29	FPAR Field Descriptions, Large Page Device (ORx[PGS] = 1).....	10-37
10-30	FBCR Field Descriptions .....	10-38
10-31	FECCn Field Descriptions .....	10-39
10-32	GPCM Read Control Signal Timing .....	10-44
10-33	GPCM Write Control Signal Timing .....	10-45
10-34	Boot Bank Field Values after Reset for GPCM as Boot Controller.....	10-53
10-35	FCM Chip-Select to First Command Timing.....	10-61
10-36	FCM Command, Address, and Write Data Timing Parameters.....	10-62
10-37	FCM Read Data Timing Parameters .....	10-65
10-38	Boot Bank Field Values after Reset for FCM as Boot Controller .....	10-66
10-39	UPM Routines Start Addresses.....	10-69
10-40	RAM Word Field Descriptions .....	10-75
10-41	MxMR Loop Field Use .....	10-79
10-42	Data Bus Drive Requirements For Read Cycles.....	10-84
10-43	FCM Register Settings for Soft Reset (ORn[PGS] = 1) .....	10-85
10-44	FCM Register Settings for Status Read (ORn[PGS] = 1).....	10-85
10-45	FCM Register Settings for ID Read (ORn[PGS] = 1) .....	10-86
10-46	FCM Register Settings for Page Read (ORn[PGS] = 1).....	10-86
10-47	FCM Register Settings for Block Erase (ORn[PGS] = 1) .....	10-87
10-48	FCM Register Settings for Page Program (ORn[PGS] = 1) .....	10-88
10-49	UPM Code for Single-Beat Read Access .....	10-89
10-50	UPM Code for Single-Beat Write Access.....	10-91
10-51	UPM Code for Burst Read Access.....	10-93
10-52	UPM Code for Refresh Cycle .....	10-95
10-53	UPM Code for Exception Cycle .....	10-97
11-1	Signal Properties .....	11-4
11-2	eSDHC Memory Map .....	11-5
11-3	DSADDR Field Descriptions.....	11-6
11-4	BLKATTR Field Descriptions .....	11-7
11-5	CMDARG Field Descriptions.....	11-8
11-6	XFERTYP Field Descriptions.....	11-8
11-7	Determination of Transfer Type .....	11-10
11-8	Relation Between Parameters and Name of Response Type .....	11-11
11-9	Response Bit Definition for Each Response Type .....	11-12
11-10	DATPORT Field Descriptions .....	11-13
11-11	PRSTAT Field Descriptions .....	11-14
11-12	PROCTL Field Descriptions .....	11-18
11-13	SYSCTL Field Descriptions .....	11-20
11-14	IRQSTAT Field Descriptions .....	11-24
11-15	Relation Between Command Timeout Error and Command Complete Status.....	11-27
11-16	Relation Between Data Timeout Error and Transfer Complete Status .....	11-27

## Tables

Table Number	Title	Page Number
11-17	Relation Between Command CRC Error and Command Timeout Error .....	11-27
11-18	IRQSTATEN Field Descriptions .....	11-28
11-19	IRQSIGEN Field Descriptions .....	11-30
11-20	AUTO12ERR Field Descriptions .....	11-32
11-21	Relationship Between Command CRC Error and Command Timeout Error for Auto CMD12 .....	11-33
11-22	HOSTCAPBLT Field Descriptions .....	11-34
11-23	WML Field Descriptions .....	11-35
11-24	FEVT Field Descriptions .....	11-36
11-25	HOSTVER Field Descriptions .....	11-37
11-26	Commands for MMC/SD/SDIO .....	11-59
11-27	EXT_CSD Access Modes .....	11-63
12-1	DMAC Register Summary .....	12-3
12-2	DMA Control Register (DMACR) Field Descriptions .....	12-4
12-3	DMAES Field Descriptions .....	12-7
12-4	DMAEEI Field Descriptions .....	12-9
12-5	DMASEEI Field Descriptions .....	12-9
12-6	DMACEEI Field Descriptions .....	12-10
12-7	DMACINT Field Descriptions .....	12-10
12-8	DMACERR Field Descriptions .....	12-11
12-9	DMASSRT Field Descriptions .....	12-12
12-10	DMACDNE Field Descriptions .....	12-12
12-11	DMAINT Field Descriptions .....	12-13
12-12	DMAERR Field Descriptions .....	12-14
12-13	DMAGPOR Field Descriptions .....	12-15
12-14	DCHPRI <sub>n</sub> Field Descriptions .....	12-16
12-15	TCD 32-Bit Memory Structure .....	12-17
12-16	TCD Word 0 (TCD <sub>n</sub> .saddr) Field Description .....	12-17
12-17	TCD Word 1 (TCD.{smod, ssize, dmod, dsize, soff}) Field Descriptions .....	12-18
12-18	TCD Word 2 (TCD.nbytes) Field Description .....	12-19
12-19	TCD Word 3 (TCD.slast) Field Descriptions .....	12-19
12-20	TCD Word 4 (TCD.daddr) Field Description .....	12-20
12-21	TCD Word 5 (TCD.{citer, doff}) Field Descriptions .....	12-21
12-22	TCD Word 6 (TCD.dlast_sga) Field Descriptions .....	12-22
12-23	TCD Word 7 (TCD.{biter, control/status}) Field Descriptions .....	12-22
13-1	USB External Signals .....	13-3
13-2	ULPI Signal Descriptions .....	13-3
13-3	USB Interface Memory Map .....	13-4
13-4	CAPLENGTH Register Field Descriptions .....	13-7
13-5	HCIVERSION Register Field Descriptions .....	13-7
13-6	HCSPARAMS Register Field Descriptions .....	13-7

# Tables

Table Number	Title	Page Number
13-7	HCCPARAMS Register Field Descriptions.....	13-8
13-8	DCIVERSION Register Field Descriptions.....	13-9
13-9	DCCPARAMS Register Field Descriptions.....	13-10
13-10	USBCMD Register Field Descriptions.....	13-11
13-11	USBSTS Register Field Descriptions.....	13-13
13-12	USBINTR Register Field Descriptions.....	13-15
13-13	FRINDEX Register Field Descriptions.....	13-17
13-14	FRINDEX N Values.....	13-17
13-15	PERIODICLISTBASE Register Field Descriptions.....	13-18
13-16	DEVICEADDR Register Field Descriptions.....	13-19
13-17	ASYNCLISTADDR Register Field Descriptions.....	13-20
13-18	ENDPOINTLISTADDR Register Field Descriptions.....	13-20
13-19	BURSTSIZE Register Field Descriptions.....	13-21
13-20	TXFILLTUNING Register Field Descriptions.....	13-22
13-21	ULPI VIEWPORT Field Descriptions.....	13-23
13-22	CONFIGFLAG Register Field Descriptions.....	13-25
13-23	PORTSC Register Field Descriptions.....	13-25
13-24	OTGSC Register Field Descriptions.....	13-30
13-25	USBMODE Register Field Descriptions.....	13-33
13-26	ENDPTSETUPSTAT Register Field Descriptions.....	13-34
13-27	ENDPTPRIME Register Field Descriptions.....	13-34
13-28	ENDPTFLUSH Register Field Descriptions.....	13-35
13-29	ENDPTSTATUS Register Field Descriptions.....	13-36
13-30	ENDPTCOMPLETE Register Field Descriptions.....	13-36
13-31	ENDPTCTRL0 Register Field Descriptions.....	13-37
13-32	ENDPTCTRL <sub>n</sub> Register Field Descriptions.....	13-38
13-33	SNOOP <sub>n</sub> Register Field Descriptions.....	13-40
13-34	AGE_CNT_THRESH Register Field Descriptions.....	13-41
13-35	PRI_CTRL Register Field Descriptions.....	13-42
13-36	SI_CTRL Register Field Descriptions.....	13-43
13-37	CONTROL Field Descriptions.....	13-44
13-38	Supported PHY Interfaces.....	13-45
13-39	Typ Field Encodings.....	13-48
13-40	Next Schedule Element Pointer.....	13-49
13-41	iTD Transaction Status and Control.....	13-50
13-42	Buffer Pointer Page 0 (Plus).....	13-51
13-43	iTD Buffer Pointer Page 1 (Plus).....	13-51
13-44	Buffer Pointer Page 2 (Plus).....	13-51
13-45	Buffer Pointer Page 3–6.....	13-52
13-46	Next Link Pointer.....	13-52
13-47	Endpoint and Transaction Translator Characteristics.....	13-53

## Tables

Table Number	Title	Page Number
13-48	Microframe Schedule Control.....	13-53
13-49	siTD Transfer Status and Control.....	13-54
13-50	siTD Buffer Pointer Page 0 (Plus) .....	13-55
13-51	siTD Buffer Pointer Page 1 (Plus) .....	13-55
13-52	siTD Back Link Pointer .....	13-55
13-53	qTD Next Element Transfer Pointer (DWord 0).....	13-57
13-54	qTD Alternate Next Element Transfer Pointer (DWord 1).....	13-57
13-55	qTD Token (DWord 2) .....	13-58
13-56	qTD Buffer Pointer .....	13-61
13-57	Queue Head DWord 0 .....	13-62
13-58	Endpoint Characteristics: Queue Head DWord 1 .....	13-63
13-59	Endpoint Capabilities: Queue Head DWord 2 .....	13-64
13-60	Current qTD Link Pointer .....	13-65
13-61	Host-Controller Rules for Bits in Overlay (DWords 5, 6, 8, and 9) .....	13-66
13-62	FTSN Normal Path Pointer .....	13-67
13-63	FSTN Back Path Link Pointer .....	13-67
13-64	Behavior During Wake-Up Events.....	13-71
13-65	Operation of FRINDEX and SOFV (SOF Value Register).....	13-75
13-66	Example Periodic Reference Patterns for Interrupt Transfers .....	13-88
13-67	Ping Control State Transition Table .....	13-89
13-68	Interrupt IN/OUT Do Complete Split State Execution Criteria.....	13-103
13-69	Initial Conditions for OUT siTD TP and T-Count Fields .....	13-112
13-70	Transaction Position (TP)/Transaction Count (T-Count) Transition Table .....	13-112
13-71	Summary siTD Split Transaction State.....	13-116
13-72	Example Case 2a—Software Scheduling siTDs for an IN Endpoint.....	13-117
13-73	Summary of Transaction Errors .....	13-120
13-74	Summary Behavior on Host System Errors .....	13-123
13-75	Endpoint Capabilities/Characteristics .....	13-125
13-76	Current dTD Pointer.....	13-126
13-77	Multiple Mode Control .....	13-127
13-78	Next dTD Pointer .....	13-127
13-79	dTD Token .....	13-128
13-80	Buffer Pointer Page 0 .....	13-128
13-81	Buffer Pointer Page 1 .....	13-129
13-82	Buffer Pointer Pages 2–4 .....	13-129
13-83	Device Controller State Information Bits .....	13-131
13-84	Device Controller Endpoint Initialization.....	13-134
13-85	Device Controller Stall Response Matrix .....	13-135
13-86	Variable Length Transfer Protocol Example (ZLT = 0).....	13-137
13-87	Variable Length Transfer Protocol Example (ZLT = 1).....	13-137
13-88	Interrupt/Bulk Endpoint Bus Response Matrix.....	13-138

# Tables

Table Number	Title	Page Number
13-89	Control Endpoint Bus Response Matrix .....	13-140
13-90	Isochronous Endpoint Bus Response Matrix .....	13-143
13-91	Device Error Matrix .....	13-148
13-92	Error Descriptions .....	13-148
13-93	Interrupt Handling Order .....	13-148
13-94	Low Frequency Interrupt Events.....	13-149
13-95	Error Interrupt Events .....	13-149
13-96	Functional Differences Between EHCI and EHCI with Embedded TT .....	13-151
13-97	Emulated Handshakes .....	13-152
13-98	ULPI Timing .....	13-156
14-1	PCI Express Interface Signals—Detailed Signal Descriptions.....	14-5
14-2	PCI Express Controller Register Groups .....	14-6
14-3	PCI Express Memory Map.....	14-6
14-4	PCI Express Vendor ID Register Field Description .....	14-15
14-5	PCI Express Device ID Register Field Description .....	14-15
14-6	PCI Express Command Register Fields Description .....	14-16
14-7	PCI Express Status Register Fields Description .....	14-17
14-8	PCI Express Revision ID Register Fields Description.....	14-18
14-9	PCI Express Class Code Register Fields Description .....	14-19
14-10	PCI Express Bus Cache Line Size Register Fields Description.....	14-19
14-11	PCI Express Latency Timer Register Fields Description .....	14-20
14-12	PCI Express Header Type Register Fields Description.....	14-20
14-13	BAR0 and BAR1 Register Fields Description.....	14-22
14-14	BAR2 and BAR4 Register Fields Description.....	14-23
14-15	BAR3 and BAR5 Register Fields Description.....	14-23
14-16	PCI Express Subsystem Vendor ID Register Fields Description .....	14-24
14-17	PCI Express Subsystem ID Register Fields Description .....	14-24
14-18	PCI Express Capabilities Pointer Register Fields Description .....	14-25
14-19	PCI Express Interrupt Line Register Fields Description.....	14-25
14-20	PCI Express MInimum Grant Register Fields Description.....	14-26
14-21	PCI Express Maximum Latency Register Fields Description .....	14-26
14-22	PCI Express Primary Bus Number Register Fields Description.....	14-27
14-23	PCI Express Secondary Bus Number Register Fields Description.....	14-28
14-24	PCI Express Subordinate Bus Number Register Fields Description .....	14-28
14-25	PCI Express I/O Base Register Fields Description .....	14-29
14-26	PCI Express I/O Limit Register Fields Description.....	14-29
14-27	PCI Express Secondary Status Register Fields Description .....	14-30
14-28	PCI Express Memory Base Register Fields Description .....	14-31
14-29	PCI Express Memory Limit Register Fields Description .....	14-31
14-30	PCI Express Prefetchable Memory Base Register Fields Description .....	14-32
14-31	PCI Express Prefetchable Memory Limit Register Fields Description .....	14-32

# Tables

<b>Table Number</b>	<b>Title</b>	<b>Page Number</b>
14-32	PCI Express Prefetchable Base Upper 32-Bit Register Fields Description .....	14-33
14-33	PCI Express Prefetchable Limit Upper 32-Bit Register Fields Description .....	14-33
14-34	PCI Express I/O Base Upper 16-Bit Register Fields Description .....	14-33
14-35	PCI Express I/O Limit Upper 16-Bit Register Fields Description .....	14-34
14-36	PCI Express Capabilities Pointer Register Fields Description .....	14-34
14-37	PCI Express Interrupt Line Register Fields Description .....	14-35
14-38	PCI Express Bridge Control Register Fields Description .....	14-35
14-39	PCI Express Power Management Capability ID Register Fields Description .....	14-37
14-40	PCI Express Power Management Next Capabilities Pointer Fields Description .....	14-37
14-41	PCI Express Power Management Capabilities Register Fields Description .....	14-38
14-42	PCI Express Power Management Status and Control Register Fields Description .....	14-38
14-43	PCI Express Power Management Data Register Fields Description .....	14-39
14-44	PCI Express Capability ID Register Fields Description .....	14-40
14-45	PCI Express Next Capabilities Pointer Fields Description .....	14-40
14-46	PCI Express Capabilities Register Fields Description .....	14-40
14-47	PCI Express Device Capabilities Register Fields Description .....	14-41
14-48	PCI Express Device Control Register Fields Description .....	14-42
14-49	PCI Express Device Status Register Fields Description .....	14-43
14-50	PCI Express Link Capabilities Register Fields Description .....	14-44
14-51	PCI Express Link Control Register Fields Description .....	14-44
14-52	PCI Express Link Status Register Fields Description .....	14-45
14-53	PCI Express Slot Capabilities Register Fields Description .....	14-46
14-54	PCI Express Slot Control Register Fields Description .....	14-46
14-55	PCI Express Slot Status Register Fields Description .....	14-47
14-56	PCI Express Root Control Register Fields Description .....	14-48
14-57	PCI Express Root Status Register Fields Description .....	14-48
14-58	PCI Express Capability ID Register Fields Description .....	14-49
14-59	PCI Express MSI Message Control Register Fields Description .....	14-49
14-60	PCI Express MSI Message Address Register Fields Description .....	14-50
14-61	PCI Express MSI Message Upper Address Register Fields Description .....	14-50
14-62	PCI Express MSI Message Data Register Fields Description .....	14-51
14-63	PCI Express Advanced Error Reporting Capability ID Register Fields Description .....	14-53
14-64	PCI Express Uncorrectable Error Status Register Fields Description .....	14-54
14-65	PCI Express Uncorrectable Error Mask Register Fields Description .....	14-55
14-66	PCI Express Uncorrectable Error Severity Register Fields Description .....	14-56
14-67	PCI Express Correctable Error Status Register Fields Description .....	14-57
14-68	PCI Express Correctable Error Mask Register Fields Description .....	14-57
14-69	PCI Express Advanced Error Capabilities and Control Register Fields Description .....	14-58
14-70	PCI Express Header Log Register Fields Description .....	14-59
14-71	PCI Express Root Error Command Register Fields Description .....	14-60
14-72	PCI Express Root Error Status Register Fields Description .....	14-61



# Tables

Table Number	Title	Page Number
14-73	PCI Express Error Source Identification Register Fields Description .....	14-61
14-74	PEX_LTSSM_STAT Fields Description .....	14-62
14-75	PEX_LTSSM_STAT Status Codes .....	14-62
14-76	PCI Express N_FTS Control Register Fields Description .....	14-64
14-77	PCI Express ACK Replay Timeout Register Fields Description .....	14-65
14-78	PEX_GCLK_RATIO Fields Description .....	14-66
14-79	PEX_PM_TIMER Fields Description .....	14-66
14-80	PEX_PME_TIMEOUT Fields Description .....	14-67
14-81	PCI Express ASPM Request Timer Register Fields Description .....	14-68
14-82	PEX_SSVID_UPDATE Fields Description .....	14-68
14-83	PCI Express Device Capabilities Update Register Fields Description .....	14-69
14-84	PCI Express Link Capabilities Update Register Fields Description .....	14-70
14-85	PCI Express Slot Capabilities Update Register Fields Description .....	14-71
14-86	PEX_CFG_READY Fields Description .....	14-72
14-87	PEX_BAR_SIZEL Fields Description .....	14-73
14-88	PEX_BAR_SEL Fields Description .....	14-73
14-89	PEX_BAR_PF Fields Description .....	14-74
14-90	PEX_PME_TO_ACK_TOR Fields Description .....	14-75
14-91	PEX_PME_TO_ACK_SR Fields Description .....	14-75
14-92	PEX_SS_INTR_MASK Fields Description .....	14-76
14-93	PEX_CSB_CTRL Register Fields Description .....	14-78
14-94	PEX_DMA_DSTMR Fields Description .....	14-78
14-95	PEX_CSB_STAT Register Fields Description .....	14-79
14-96	PEX_CSB_OBCTRL Register Fields Description .....	14-80
14-97	PEX_CSB_OBSTAT Register Fields Description .....	14-81
14-98	PEX_CSB_IBCTRL Register Fields Description .....	14-82
14-99	PEX_CSB_IBSTAT Register Fields Description .....	14-82
14-100	PEX_WDMA_CTRL Register Fields Description .....	14-83
14-101	PEX_WDMA_ADDR Register Fields Description .....	14-84
14-102	PEX_WDMA_STAT Register Fields Description .....	14-85
14-103	PEX_RDMA_CTRL Register Fields Description .....	14-85
14-104	PEX_RDMA_ADDR Register Fields Description .....	14-86
14-105	PEX_RDMA_STAT Register Fields Description .....	14-87
14-106	PEX_OMBCR Register Fields Description .....	14-88
14-107	PEX_OMBDR Register Fields Description .....	14-88
14-108	PEX_IMBCR Register Fields Description .....	14-88
14-109	PEX_IMBDR Register Fields Description .....	14-89
14-110	PEX_HIER Register Fields Description .....	14-90
14-111	PEX_HISR Register Fields Description .....	14-91
14-112	PEX_HOPIVR Register Fields Description .....	14-92
14-113	PEX_HIPIVR Register Fields Description .....	14-92

# Tables

Table Number	Title	Page Number
14-114	PEX_HWDIVR Register Fields Description .....	14-93
14-115	PEX_HRDIVR Register Fields Description .....	14-93
14-116	PEX_HMIVR Register Fields Description .....	14-93
14-117	PEX_CSPIER Register Fields Description .....	14-94
14-118	PEX_CSWDIER Register Fields Description .....	14-95
14-119	PEX_CSRDIER Register Fields Description .....	14-96
14-120	PEX_CSMIER Register Fields Description .....	14-96
14-121	PEX_CSPISR Register Fields Description .....	14-98
14-122	PEX_CSWDISR Register Fields Description .....	14-99
14-123	PEX_CSRDISR Register Fields Description .....	14-100
14-124	PEX_CSMISR Register Fields Description .....	14-100
14-125	PEX_PM_CTRL Register Fields Description .....	14-102
14-126	PEX_OWAR0–PEX_OWAR3 Register Fields Description .....	14-103
14-127	PEX_OWBAR <sub>n</sub> Register Fields Description .....	14-105
14-128	PEX_OWSTARL <sub>n</sub> Register Fields Description .....	14-105
14-129	PEX_OWSTARH <sub>n</sub> Register Fields Description .....	14-106
14-130	EP Inbound Base and Translation Address Registers Correspondence .....	14-106
14-131	PEX_EPIWTAR <sub>n</sub> Register Fields Description .....	14-107
14-132	PEX_RCIWAR <sub>n</sub> Register Fields Description .....	14-108
14-133	PEX_RCIWTAR <sub>n</sub> Register Fields Description .....	14-109
14-134	PEX_RCIWBARL <sub>n</sub> Register Fields Description .....	14-110
14-135	PEX_RCIWBARH <sub>n</sub> Register Fields Description .....	14-110
14-136	Address Translation Window Combinations .....	14-112
14-137	PCI Express Transactions .....	14-113
14-138	Configuration Address Mapping .....	14-116
14-139	PCI Express RC Inbound Message Handling .....	14-120
14-140	PCI Express EP Inbound Message Handling .....	14-121
14-141	Initial Credit Advertisement .....	14-123
14-142	Power Management State Supported .....	14-125
14-143	DMA Descriptor Bit Fields Description .....	14-128
15-1	SerDes External Signals—Detailed Signal Descriptions .....	15-2
15-2	SerDes PHY Block Memory Map .....	15-3
15-3	SRDSCR0 Field Descriptions .....	15-4
15-4	SRDSCR1 Field Descriptions .....	15-6
15-5	SRDSCR2 Field Descriptions .....	15-7
15-6	SRDSCR3 Field Descriptions .....	15-8
15-7	SRDSCR4 Field Descriptions .....	15-9
15-8	SRDSRSTCTL Field Descriptions .....	15-10
16-1	eTSEC <sub>n</sub> Network Interface Signal Properties .....	16-6
16-2	eTSEC Signals—Detailed Signal Descriptions .....	16-7
16-3	Module Memory Map Summary .....	16-10

# Tables

Table Number	Title	Page Number
16-4	Module Memory Map .....	16-11
16-5	TSEC_ID Field Descriptions .....	16-21
16-6	TSEC_ID2 Field Descriptions .....	16-22
16-7	TSEC_ID2[TSEC_INT] Field Settings .....	16-22
16-8	IEVENT Field Descriptions .....	16-24
16-9	IMASK Field Descriptions .....	16-27
16-10	EDIS Field Descriptions .....	16-29
16-11	ECNTRL Field Descriptions.....	16-30
16-12	eTSEC Interface Configurations .....	16-31
16-13	PTV Field Descriptions .....	16-32
16-14	DMACTRL Field Descriptions.....	16-32
16-15	TBIPA Field Descriptions .....	16-34
16-16	TCTRL Field Descriptions.....	16-34
16-17	TSTAT Field Descriptions.....	16-37
16-18	DFVLAN Field Descriptions .....	16-40
16-19	TXIC Field Descriptions .....	16-41
16-20	TQUEUE Field Descriptions .....	16-42
16-21	TR03WT Field Descriptions .....	16-43
16-22	TR47WT Field Descriptions .....	16-44
16-23	TBPTR <sub>n</sub> Field Descriptions .....	16-44
16-24	TBASE0–TBASE7 Field Descriptions .....	16-45
16-25	TMR_TXTSn_ID Register Field Descriptions .....	16-45
16-26	TMR_TXTSn_H/L Register Field Descriptions.....	16-46
16-27	RCTRL Field Descriptions .....	16-47
16-28	RSTAT Field Descriptions .....	16-49
16-29	RXIC Field Descriptions.....	16-51
16-30	RQUEUE Field Descriptions .....	16-51
16-31	RBIFX Field Descriptions .....	16-52
16-32	RQFAR Field Descriptions .....	16-54
16-33	RQFCR Field Descriptions .....	16-54
16-34	RQFPR Field Descriptions.....	16-56
16-35	MRBLR Field Descriptions .....	16-59
16-36	RBPTR <sub>n</sub> Field Descriptions.....	16-60
16-37	RBASE0–RBASE7 Field Descriptions .....	16-61
16-38	TMR_RXTS_H/L Register Field Descriptions.....	16-61
16-39	MACCFG1 Field Descriptions .....	16-64
16-40	MACCFG2 Field Descriptions .....	16-66
16-41	IPGIFG Field Descriptions .....	16-68
16-42	HAFDUP Field Descriptions .....	16-69
16-43	MAXFRM Field Descriptions .....	16-70
16-44	MIIMCFG Field Descriptions.....	16-70

## Tables

Table Number	Title	Page Number
16-45	MIIMCOM Descriptions.....	16-71
16-46	MIIMADD Field Descriptions.....	16-72
16-47	MIIMCON Field Descriptions.....	16-72
16-48	MIIMSTAT Field Descriptions.....	16-73
16-49	MIIMIND Field Descriptions.....	16-73
16-50	IFSTAT Field Descriptions.....	16-74
16-51	MACSTNADDR1 Field Descriptions.....	16-74
16-52	MACSTNADDR2 Field Descriptions.....	16-75
16-53	MAC <sub>n</sub> ADDR1 Field Descriptions.....	16-76
16-54	MAC01ADDR2–MAC15ADDR2 Field Descriptions.....	16-77
16-55	TR64 Field Descriptions.....	16-78
16-56	TR127 Field Descriptions.....	16-78
16-57	TR255 Field Descriptions.....	16-79
16-58	TR511 Field Descriptions.....	16-79
16-59	TR1K Field Descriptions.....	16-80
16-60	TRMAX Field Descriptions.....	16-80
16-61	TRMGV Field Descriptions.....	16-81
16-62	RBYT Field Descriptions.....	16-81
16-63	RPKT Field Descriptions.....	16-82
16-64	RFCS Field Descriptions.....	16-82
16-65	RMCA Field Descriptions.....	16-82
16-66	RBCA Field Descriptions.....	16-83
16-67	RXCF Field Descriptions.....	16-83
16-68	RXPF Field Descriptions.....	16-84
16-69	RXUO Field Descriptions.....	16-84
16-70	RALN Field Descriptions.....	16-85
16-71	RFLR Field Descriptions.....	16-85
16-72	RCDE Field Descriptions.....	16-86
16-73	RCSE Field Descriptions.....	16-86
16-74	RUND Field Descriptions.....	16-87
16-75	ROVR Field Descriptions.....	16-87
16-76	RFRG Field Descriptions.....	16-88
16-77	RJBR Field Descriptions.....	16-88
16-78	RDRP Field Descriptions.....	16-89
16-79	TBYT Field Descriptions.....	16-89
16-80	TPKT Field Descriptions.....	16-90
16-81	TMCA Field Descriptions.....	16-90
16-82	TBCA Field Descriptions.....	16-91
16-83	TXPF Field Descriptions.....	16-91
16-84	TDFR Field Descriptions.....	16-92
16-85	TEDF Field Descriptions.....	16-92

# Tables

Table Number	Title	Page Number
16-86	TSCL Field Descriptions .....	16-93
16-87	TMCL Field Descriptions .....	16-93
16-88	TLCL Field Descriptions .....	16-94
16-89	TXCL Field Descriptions .....	16-94
16-90	TNCL Field Descriptions .....	16-95
16-91	TDRP Field Descriptions .....	16-95
16-92	TJBR Field Descriptions .....	16-96
16-93	TFCS Field Descriptions .....	16-96
16-94	TXCF Field Descriptions .....	16-97
16-95	TOVR Field Descriptions .....	16-97
16-96	TUND Field Descriptions .....	16-98
16-97	TFRG Field Descriptions .....	16-98
16-98	CAR1 Field Descriptions .....	16-99
16-99	CAR2 Field Descriptions .....	16-100
16-100	CAM1 Field Descriptions .....	16-101
16-101	CAM2 Field Descriptions .....	16-103
16-102	RREJ Field Descriptions .....	16-104
16-103	IGADDR <sub>n</sub> Field Descriptions .....	16-105
16-104	GADDR <sub>n</sub> Field Descriptions .....	16-106
16-105	ATTR Field Descriptions .....	16-106
16-106	ATTRELI Field Descriptions .....	16-107
16-107	RQPRM Field Descriptions .....	16-108
16-108	RFBPTR0–RFBPTR7 Field Descriptions .....	16-109
16-109	TMR_CTRL Register Field Descriptions .....	16-110
16-110	TMR_TEVENT Register Field Descriptions .....	16-112
16-111	TMR_TEMASK Register Definition .....	16-113
16-112	TMR_TEMASK Register Field Descriptions .....	16-113
16-113	TMR_PEVENT Register Field Descriptions .....	16-114
16-114	TMR_PEMASK Register Field Descriptions .....	16-115
16-115	TMR_STAT Register Definition .....	16-115
16-116	TMR_STAT Register Field Descriptions .....	16-115
16-117	TMR_CNT_H/L Register Field Descriptions .....	16-116
16-118	TMR_ADD Register Field Descriptions .....	16-117
16-119	TMR_ACC Register Field Descriptions .....	16-117
16-120	TMR_PRSC Register Field Descriptions .....	16-118
16-121	TMROFF_H/L Register Field Descriptions .....	16-118
16-122	TMR_ALARM <sub>n</sub> _H/L Register Field Descriptions .....	16-119
16-123	TMR_FIPER Register Field Descriptions .....	16-120
16-124	TMR_ETTS1-2_H Register Field Descriptions .....	16-120
16-125	RGMI and MII Signals Multiplexing .....	16-123
16-126	Shared Signals .....	16-123

## Tables

<b>Table Number</b>	<b>Title</b>	<b>Page Number</b>
16-127	Steps for Minimum Register Initialization.....	16-124
16-128	Custom Preamble Field Descriptions.....	16-129
16-129	Received Preamble Field Descriptions .....	16-130
16-130	Flow Control Frame Structure .....	16-134
16-131	Non-Error Transmit Interrupts .....	16-135
16-132	Non-Error Receive Interrupts.....	16-135
16-133	Interrupt Coalescing Timing Threshold Ranges .....	16-137
16-134	Transmission Errors .....	16-138
16-135	Reception Errors .....	16-138
16-136	Tx Frame Control Block Description.....	16-141
16-137	Rx Frame Control Block Descriptions.....	16-143
16-138	Supported Stack L2 Ethernet Headers .....	16-145
16-139	Special Filer Rules .....	16-149
16-140	Receive Queue Filer Interrupt Events.....	16-149
16-141	Filer Table Example—802.1p Priority Filing .....	16-150
16-142	Filer Table Example—IP Diff-Serv Code Points Filing .....	16-151
16-143	Filer Table Example—TCP and UDP Port Filing.....	16-152
16-144	PTP Payload Special Fields.....	16-159
16-145	Time-Stamp Insertion Programming Requirements .....	16-161
16-146	Tx Frame Control Block Description.....	16-163
16-147	Transmit Data Buffer Descriptor (TxBD) Field Descriptions .....	16-167
16-148	Receive Buffer Descriptor Field Descriptions .....	16-170
16-149	MII Interface Mode Signal Configuration .....	16-172
16-150	Shared MII Signals.....	16-173
16-151	MII Mode Register Initialization Steps.....	16-173
16-152	RGMI I Interface Mode Signal Configuration.....	16-175
16-153	Shared RGMII Signals .....	16-176
16-154	RGMI I Mode Register Initialization Steps .....	16-176
17-1	I <sup>2</sup> C Interface Signal Descriptions .....	17-3
17-2	I <sup>2</sup> C Interface Signals—Detailed Signal Descriptions .....	17-4
17-3	I <sup>2</sup> C Memory Map .....	17-4
17-4	I2CADR Field Descriptions .....	17-5
17-5	I2C FDR Field Descriptions .....	17-6
17-6	I2CCR Field Descriptions .....	17-7
17-7	I2CSR Field Descriptions .....	17-8
17-8	I2CDR Field Descriptions.....	17-9
17-9	I2CDFSRR Field Descriptions.....	17-10
18-1	DUART Signal Overview .....	18-3
18-2	DUART Signals—Detailed Signal Descriptions .....	18-3
18-3	DUART Register Summary .....	18-4
18-4	URBR Field Descriptions .....	18-5

# Tables

Table Number	Title	Page Number
18-5	UTHR Field Descriptions .....	18-6
18-6	UDMB Field Descriptions .....	18-6
18-7	UDLB Field Descriptions .....	18-7
18-8	Baud Rate Examples .....	18-7
18-9	UIER Field Descriptions .....	18-8
18-10	UIIR Field Descriptions .....	18-9
18-11	UIIR IID Bits Summary .....	18-9
18-12	UFCR Field Descriptions .....	18-10
18-13	ULCR Field Descriptions .....	18-11
18-14	Parity Selection Using ULCR[PEN], ULCR[SP], and ULCR[EPS] .....	18-12
18-15	UMCR Field Descriptions .....	18-12
18-16	ULSR Field Descriptions .....	18-13
18-17	USCR Field Descriptions .....	18-14
18-18	UAFR Field Descriptions .....	18-15
18-19	UDSR Field Descriptions .....	18-15
18-20	UDSR[TXRDY] Set Conditions .....	18-16
18-21	UDSR[TXRDY] Cleared Conditions .....	18-16
18-22	UDSR[RXRDY] Set Conditions .....	18-16
18-23	UDSR[RXRDY] Cleared .....	18-16
19-1	Signal Properties .....	19-6
19-2	Detailed Signal Descriptions .....	19-6
19-3	SPI Register Summary .....	19-8
19-4	SPMODE Field Descriptions .....	19-8
19-5	SPIE Field Descriptions .....	19-11
19-6	SPIM Field Descriptions .....	19-12
19-7	SPCOM Field Descriptions .....	19-13
19-8	SPI Transmit Data Hold Field Descriptions .....	19-13
19-9	SPI Receive Data Hold Field Descriptions .....	19-14
20-1	JTAG Test Signals Summary .....	20-2
20-2	JTAG Test—Detailed Signal Descriptions .....	20-2
21-1	GPIO—Signal Descriptions .....	21-2
21-2	GPIO Register Address Map .....	21-2
21-3	GPDIR Bit Settings .....	21-3
21-4	GPODR Bit Settings .....	21-3
21-5	GPnDAT Bit Settings .....	21-4
21-6	GPIER Bit Settings .....	21-4
21-7	GPIMR Bit Settings .....	21-5
21-8	GPICR Bit Settings .....	21-5
A-1	Local Access Register Memory Map .....	A-1
A-2	System Configuration Registers .....	A-2
A-3	Watchdog Timer (WDT) Registers .....	A-3

# Tables

<b>Table Number</b>	<b>Title</b>	<b>Page Number</b>
A-4	Real Time Clock (RTC) Registers .....	A-3
A-5	Periodic Interval Timer (PIT) Registers.....	A-3
A-6	General Purpose (Global) Timers (GTM) Registers.....	A-4
A-7	IPIC Registers .....	A-5
A-8	System Arbiter Registers .....	A-6
A-9	Reset Configuration Registers .....	A-6
A-10	Clock Configuration Registers.....	A-7
A-11	Power Management Controller (PMC) Registers .....	A-7
A-12	General Purpose I/O (GPIO) Registers.....	A-7
A-13	DDR Memory Controller Registers .....	A-8
A-14	I <sup>2</sup> C Controller Registers.....	A-9
A-15	DUART Registers .....	A-9
A-16	Enhanced Local Bus Controller Registers .....	A-11
A-17	Serial Peripheral Interface (SPI) Registers .....	A-12
A-18	DMA Controller Registers .....	A-13
A-19	Registers .....	A-14
A-20	PCI Express Controller Registers .....	A-14
A-21	Enhanced Three-Speed Ethernet Controllers (eTSECs) Registers .....	A-21
A-22	SerDes PHY Registers .....	A-31
A-23	Enhanced Secure Digital Host Controller (eSDHC) Registers.....	A-32
A-24	USB Interface Registers.....	A-32



## About This Book

This reference manual defines the functionality of the MPC8308. The device is a cost-effective, low-power, highly integrated host processor that addresses the requirements of networking applications such as low-end printing, smart grid home energy gateway, data concentrators, wireless LAN access points, wireless femto base stations, and industrial applications, such as industrial control and factory automation.

The MPC8308 extends the PowerQUICC II Pro family, adding high CPU performance, additional functionality, and faster interfaces while addressing the requirements related to time-to-market, price, power consumption, and package size.

## Audience

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

## Organization

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

- [Chapter 1, “Overview,”](#) provides a high-level description of features and functionality of the MPC8308. It describes the device, its interfaces, and the programming model. The functional operation of the device, with emphasis on peripheral functions, is also described.
- [Chapter 2, “Signal Descriptions,”](#) provides a listing of all the external signals, cross-references for signals that serve multiple functions, their functional blocks, and I/O states. Also, these signals are listed by alphabetical order.
- [Chapter 3, “Memory Map,”](#) describes the memory map of the device. An overview of the local address map is provided. Next, a complete listing of all memory-mapped registers is provided, with cross references to the sections detailing descriptions of each.
- [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 device.
- [Chapter 5, “System Configuration,”](#) provides an overview of several functions that control the local access windows, system configuration, software watchdog, real time clock, periodic and general purpose timers, power management, protection, and general utilities.
- [Chapter 6, “Arbiter and Bus Monitor,”](#) provides an overview of the arbiter in the device. Also, it describes the configuration, control, and status registers of the arbiter.
- [Chapter 7, “e300 Processor Core Overview,”](#) provides an overview of the basic functionality of the processor core and briefly describes how the functional units interact.
- [Chapter 8, “Integrated Programmable Interrupt Controller \(IPIC\),”](#) describes the IPIC interrupt protocol, various types of interrupt sources controlled by the IPIC unit, and the IPIC registers with

some programming guidelines. It also provides a definition of the external interrupt signals and their functions. In addition, the interrupt configuration, control, and status registers are described in this chapter.

- **Chapter 9, “DDR Memory Controller,”** describes the DDR2 memory controller of the device. This fully programmable controller supports most DDR memories available today, including both buffered and unbuffered devices. Dynamic power management and auto-precharge modes simplify memory system design.
- **Chapter 10, “Enhanced Local Bus Controller,”** describes the enhanced local bus controller (eLBC) of the device. It describes the external signals and the memory-mapped registers as well as a functional description of the general-purpose chip-select machine (GPCM), Flash control machine (FCM), and user-programmable machines (UPMs) of the eLBC. Also, it includes an initialization and applications information section with many specific examples of its use.
- **Chapter 11, “Enhanced Secure Digital Host Controller,”** describes the enhanced SD Host Controller, which provides an interface between the host system and SD/MMC/SDIO cards. It provides a functional description of the major system blocks and includes command information for the host.
- **Chapter 12, “DMA Controller (DMAC),”** describes a second-generation platform module capable of performing complex data transfers with minimal intervention from a host processor using  $n$  programmable channels. It is intended for use in applications where the data size to be transferred is statically known, and is not defined within the data packet itself. The DMA hardware supports single design with two channels (Tx and Rx), 32-byte transfer control descriptor per channel stored in local memory, and 32 bytes of data registers, used as temporary storage to support burst transfers.
- **Chapter 13, “Universal Serial Bus Interface,”** describes the universal serial bus (USB) interface. The USB DR module is a USB 2.0-compliant serial interface engine for implementing a USB interface. The DR module supports the required signaling for UTMI low pin count interface (ULPI) transceivers (PHYs). An external PHY would be used to interface to ULPI.
- **Chapter 14, “PCI Express Interface Controller,”** describes the PCI Express interface controller, which connects the CSB to the PCI Express bus, a 2.5 GHz serial interface that supports up to a x2 lane. As both a master (initiator) and a target device, the PCI Express interface is capable of high bandwidth data transfer and is designed to support the next generation I/O devices.
- **Chapter 15, “SerDes PHY,”** describes the block which includes the serializer/deserializer PHY, the protocol converter per protocol, the protocol mux, and the control registers and control logic. It supports x1 PCI Express.
- **Chapter 16, “Enhanced Three-Speed Ethernet Controllers,”** describes the two enhanced three-speed Ethernet controllers on the device. These controllers provide 10/100/1Gb Ethernet support with a set of media-independent interface options including MII and RGMII. The controllers provide two full-duplex FIFO interface modes and quality of service support.
- **Chapter 17, “I<sup>2</sup>C Interface,”** describes the inter-IC (IIC or I<sup>2</sup>C) bus controllers of the device. These synchronous, serial, bidirectional, multiple-master buses allow two-wire connection of devices, such as microcontrollers, EEPROMs, real-time clock devices, A/D converters, and LCDs. The device powers up in boot sequencer mode, which allows the I<sup>2</sup>C controllers to initialize configuration registers.

- [Chapter 18, “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 19, “Serial Peripheral Interface,”](#) describes the MPC8308 serial peripheral interface (SPI) that allows the exchange of data between the MPC8308 and MPC83xx family of 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 20, “JTAG/Testing Support,”](#) describes the joint test action group (JTAG) interface of the MPC8308 to facilitate boundary-scan testing. The JTAG interface complies to the IEEE 1149.1™ boundary-scan specification.
- [Chapter 21, “General Purpose I/O \(GPIO\),”](#) describes the general purpose I/O (GPIO) module in the MPC8308, including a definition of the external signals and functions they serve. Additionally, interrupt capabilities, pin description, and register settings are described.
- [Appendix A, “Complete List of Configuration, Control, and Status Registers,”](#) lists all the registers used with the MPC8308.

## 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


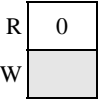
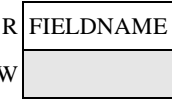

Freescal documentation is available from the sources listed on the back cover of this manual:

- *e300 Core Reference Manual*—This book provides a more detailed description of the e300 core.
- *MPC8308 PowerQUICC II Pro Processor Device Errata*—This document details all known silicon errata for the Freescale IPs in the MPC8308.
- *MPC8308 PowerQUICC II Pro Processor Hardware Specification*—This document provides an overview of the MPC8308 features, its hardware specifications, including a block diagram showing the major functional components. Hardware specifications provide specific data regarding bus timing, signal behavior, and AC, DC, and thermal characteristics, as well as other design considerations.

For more information on other device 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.
<b>mnemonics</b>	Instruction mnemonics are shown in lowercase bold
<i>italics</i>	Italics indicate variable command parameters, for example, <b>bcctrx</b> Book titles in text are set in italics Internal signals are set in lowercase italics, for example, <u>core_int</u>
0x0	Prefix to denote hexadecimal number
0b0	Prefix to denote binary number
rA, rB	Instruction syntax used to identify a source GPR
rD	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
<i>x</i>	An italicized <i>x</i> indicates an alphanumeric variable
<i>n</i>	An italicized <i>n</i> 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 register. Although these bits can be written to as ones or zeros, they are always read as zeros.
	Indicates a reserved bit field in a memory-mapped register. Although these bits can be written to as ones or zeros, they are always read as zeros.
	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

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.

## Acronyms and Abbreviations

Table i contains acronyms and abbreviations used in this document.

**Table i. Acronyms and Abbreviated Terms**

Term	Meaning
AFEU	ARC four execution unit
BD	Buffer descriptor
BIST	Built-in self test
CD	Collision detect
COL	Collision
CPM	Communication processor module
CRC	Cyclic redundancy check
CRS	Carrier sense
CSB	Coherent system bus
CSMA	Carrier-sense multiple access
DDR	Double data rate
DMA	Direct memory access
DRAM	Dynamic random access memory
DTLB	Data translation lookaside buffers
DUART	Dual universal asynchronous receiver/transmitter
EA	Effective address
ECC	Error checking and correction
EHCI	Enhanced host controller interface
EHPI	Enhanced host port interface
EPROM	Erasable programmable read-only memory
FS	Full-speed
FCS	Frame-check sequence
GMII	Gigabit media independent interface
GPCM	General-purpose chip-select machine
GPIO	General-purpose I/O
GPR	General-purpose register
GTM	General purpose timers
IAD	Internet access device

**Table i. Acronyms and Abbreviated Terms (continued)**

<b>Term</b>	<b>Meaning</b>
I <sup>2</sup> C	Inter-integrated circuit
IEEE	Institute of Electrical and Electronics Engineers
IOS	I/O sequencer
IPG	Interpacket gap
ISDN	Integrated services digital network
ITLB	Instruction translation lookaside buffer
IU	Integer unit
JTAG	Joint Test Action Group
LALE	LBC external address latch enable
LBC	Local bus controller
LRU	Least recently used
LSB	Least-significant byte
lsb	Least-significant bit
LSU	Load/store unit
MAC	Multiply accumulate, media access control
MCP	Machine-check interrupt
MDI	Medium-dependent interface
MDEU	Message digest execution unit
MIB	Management information base
MII	Media independent interface
MMU	Memory management unit
MPH	Multi-port host
MSB	Most-significant byte
msb	Most-significant bit
OSI	Open systems interconnection
PCI	Peripheral component interconnect
PCS	Physical coding sublayer
PIC	Programmable interrupt controller
PIT	Periodic interval timer
PKEU	Public key execution unit
PMA	Physical medium attachment
PMD	Physical medium dependent
POR	Power-on reset

**Table i. Acronyms and Abbreviated Terms (continued)**

<b>Term</b>	<b>Meaning</b>
PRI	Primary rate interface
RGMII	Reduced gigabit media independent interface
RISC	Reduced instruction set computing
RMON	Remote monitoring
RMW	Read-modify-write
RNG	Random number generator
RTBI	Reduced ten-bit interface
RTC	Real time clock module
Rx	Receive
RxBD	Receive buffer descriptor
SCL	Serial clock
SDA	Serial data
SFD	Start frame delimiter
SI	Serial interface
SPI	Serial peripheral interface
SPR	Special-purpose register
SRAM	Static random access memory
TAP	Test access port
TBI	Ten-bit interface
TLB	Translation lookaside buffer
TSEC	Three-speed Ethernet controller
Tx	Transmit
TxBD	Transmit buffer descriptor
UART	Universal asynchronous receiver/transmitter
UPM	User-programmable machine
UTP	Unshielded twisted pair
WDT	Watchdog timer
ZBT	Zero bus turnaround





# Chapter 1

## Overview

This document provides an overview of the MPC8308 PowerQUICC II Pro processor features, including a block diagram showing the major functional components. MPC8308 is a cost-effective, low-power, highly integrated host processor. The device extends the PowerQUICC family, adding higher CPU performance, additional functionality, and faster interfaces while addressing the requirements related to time-to-market, price, power consumption, and package size.

### 1.1 MPC8308 Overview

Figure 1-1 shows the major functional units within the MPC8308. The Power™ e300 core in the MPC8308, with its 16 Kbytes of instruction and 16 Kbytes of data cache, implements the PowerPC user instruction set architecture and provides hardware and software debugging support. In addition, the MPC8308 offers a PCI Express controller, two three-speed 10, 100, 1000 Mbps Ethernet controllers (eTSECs), a DDR2 SDRAM memory controller, a SerDes block, an enhanced secure digital host controller (eSDHC), an enhanced local bus controller (eLBC), an integrated programmable interrupt controller (IPIC), a general purpose DMA controller, two I<sup>2</sup>C controllers, dual UART (DUART), GPIOs, USB, general purpose timers, and an SPI controller. The high level of integration in the MPC8308 helps simplify board design and offers significant bandwidth and performance.

A block diagram of the MPC8308 is shown in Figure 1-1.

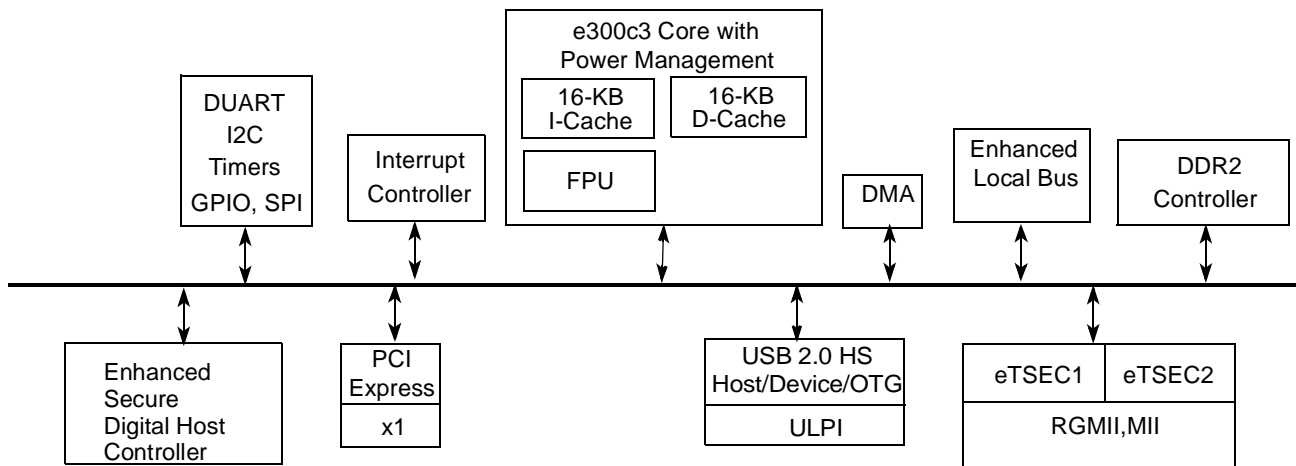


Figure 1-1. MPC8308 Block Diagram

The major features of this device are as follows:

- e300c3 processor core

- Enhanced version of the MPC603e core
- High-performance, superscalar processor core with a four-stage pipeline and low interrupt latency times
- Floating-point, dual integer units, load/store, system register, and branch processing units
- 16-Kbyte instruction cache and 16-Kbyte data cache with lockable capabilities
- Capable of completing two MACs every three cycles
- Dynamic power management
- Enhanced hardware program debug features
- Software-compatible with Freescale processor families implementing Power Architecture technology
- Separate PLL that is clocked by the system bus clock
- Performance monitor
- DDR SDRAM memory controller
  - Programmable timing supporting DDR2 SDRAM
  - Integrated SDRAM clock generation
  - Supports 8-bit ECC
  - 16-/32-bit data interface, up to 266-MHz data rate
  - 512-Mbyte addressable space for 32-bit data interface; 256-Mbyte for 16-bit data interface
  - The following SDRAM configurations are supported:
    - Up to two physical banks (chip selects), each bank up to 1 Gbyte independently addressable
    - 64-Mbit to 2-Gbit devices with  $\times 8/\times 16$  data ports (no direct  $\times 4$  support)
    - One 16-bit device or two 8-bit devices on a 16-bit bus, or two 16-bit devices or four 8-bit devices on a 32-bit bus
  - Support for up to 16 pages for DDR2
  - Two chip selects
  - Supports auto refresh
  - On-the-fly power management using CKE
  - Registered DIMM support
  - 1.8-V SSTL\_18 compatible I/O for DDR2
- Two enhanced three-speed Ethernet controllers (eTSEC)
  - Three-speed support (10/100/1000 Mbps)
  - MII/RGMII interface
  - Controllers designed to comply with IEEE Std 802.3<sup>®</sup>, 802.3u<sup>®</sup>, 802.3x<sup>®</sup>, 802.3z<sup>®</sup>, 802.3ac<sup>®</sup>, and 802.3ab<sup>®</sup>
  - TCP/IP acceleration and QoS features available
    - 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
- 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)
- Programmable maximum frame length supports jumbo frames (up to 9.6 Kbytes) and IEEE Std 802.1Q 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
- Retransmission following a collision
- 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
  - Up to 16 exact-match MAC addresses supported
  - Broadcast address (accept/reject)
  - Hash table match on up to 512 multicast addresses
  - Promiscuous mode
  - 10K packet buffers to enable check-summing for jumbo packets
- Buffer descriptors backward compatible with MPC8260 and MPC860T 10/100 Ethernet programming models
- RMON statistics support
- MII management interface for control and status
- SerDes block with one lane
  - Support for one  $\times 1$  PCI Express controller
  - Link-layer interfaces to PCI Express controller
  - SerDes power-down/reset state machine for cold (power-on) or warm (software-initiated) reset of SerDes, PHY, and controllers
- PCI Express
  - Supports one interface supporting  $\times 1$  width
  - Compatible with the *PCI Express 1.0a Specification*
  - Selectable operation as root complex or endpoint
  - 32- and 64-bit addressing

- 128-byte maximum payload size
- Virtual channel 0 only
- Traffic class 0–7
- Full 64-bit decode with 32-bit wide windows
- Four outbound translation address windows
  - Support for mapping 32-bit internal local memory space to an external 32- or 64-bit address space and translating that address within the PCI Express space
- Four inbound translation address windows corresponding to defined PCI Express BARs
  - The first BAR is 32-bits can be programmed to use on-chip register access
  - The second BAR is 32-bits, which is for general use
  - The remaining two BARs may be 32- or 64-bits and are also for general use
- Enhanced local bus controller (eLBC)
  - Non-multiplexed 26-bit address and 8-/16-bit data operating at up to 66 MHz
  - Four chip selects supporting four external slaves
  - Variable memory block sizes (32 Kbytes to 4 Gbytes in FCM mode, 32 Kbytes to 64 Mbytes in UPM mode, and 32 Kbytes to 64 Mbytes in GPCM mode)
  - Supports boot from NOR Flash and NAND Flash
  - Supports programmable clock ratio dividers
  - Up to eight-beat burst transfers
  - 16- and 8-bit ports
  - Three protocol engines available on a per-chip select basis:
    - General-purpose chip select machine (GPCM)
    - Three user programmable machines (UPMs)
    - NAND Flash control machine (FCM)
  - Default boot ROM chip select with configurable bus width (8 or 16)
  - Provides two Write Enable signals to allow single-byte write access to external 16-bit eLBC slave devices
- Integrated programmable interrupt controller (IPIC)
  - Functional and programming compatibility with the MPC8260 interrupt controller
  - Support for external and internal discrete interrupt sources
  - Programmable highest priority request
  - Six groups of interrupts with programmable priority
  - External and internal interrupts directed to host processor
  - Supports MSI functionality for PCI Express
  - Unique vector number for each interrupt source
- Two I<sup>2</sup>C interfaces
  - Two-wire interface
  - Multiple-master support

- Master or slave I<sup>2</sup>C mode support
- On-chip digital filtering rejects spikes on the bus
- General purpose DMA engine
  - Support for the DMA engine with the following features:
    - Four DMA channels
    - All data movement via dual-address transfers: read from source, write to destination
    - Transfer control descriptor (TCD) organized to support two-deep, nested transfer operations
    - Channel activation using one of two methods (for both the methods, one activation per execution of the minor loop is required):
      - Explicit software initiation
      - Initiation via a channel-to-channel linking mechanism for continuous transfers (independent channel linking at end of minor loop and/or major loop)
    - Support for fixed-priority and round-robin channel arbitration
    - Channel completion reported via optional interrupt requests
  - Support for scatter/gather DMA processing
- DUART
  - Two 2-wire interfaces (RxD and TxD)
  - Programming model compatible with the original 16450 UART and the PC16550D
- Serial peripheral interface (SPI)
  - Master or slave support
- System timers
  - Periodic interrupt timer
  - Software watchdog timer
  - Four general-purpose timers
- Enhanced secure digital host controller (eSDHC)
  - Conforms to the *SD Host Controller Standard Specification Version 2.0* including test event register support
  - Compatible with the *MMC System Specification Version 4.0*
  - Compatible with the *SD Memory Card Specification Version 2.0* and supports the high-capacity SD memory card
  - Compatible with the *SD Card Specification, Part E1, SD Input/Output (SDIO) Card Specification, Version 2.0*
  - Designed to work with SD Memory, miniSD Memory, SDIO, miniSDIO, SD Combo, MMC, MMC*plus*, and RS-MMC cards
  - Card bus clock frequency up to 50 MHz
  - Supports 1-/4-bit SD and SDIO modes,
    - Up to 200 Mbps of data transfer for SD/SDIO/MMC cards using four parallel data lines
  - Supports single- and multi-block read and write

- Supports block sizes of 1 ~ 4096 bytes
- Supports the write protection switch for write operations
- Supports synchronous abort
- Supports pause during the data transfer at block gap
- Supports SDIO read wait and suspend resume operations
- Supports Auto CMD12 for multi-block transfer
- Host can initiate non-data transfer command while data transfer is in progress
- Allows cards to interrupt the host in 1 and 4-bit SDIO modes
- Embodies a fully configurable  $128 \times 32$ -bit FIFO for read/write data
- Supports DMA capabilities
- Universal serial bus (USB) dual-role controller
  - Designed to comply with *Universal Serial Bus Revision 2.0 Specification*
  - Supports operation as a stand-alone USB host controller
    - Supports USB root hub with one downstream-facing port
    - Enhanced host controller interface (EHCI) compatible
  - Supports operation as a stand-alone USB device
    - Supports one upstream-facing port
    - Supports three programmable bidirectional USB endpoints
  - Supports high-speed (480-Mbps), full-speed (12-Mbps), and low-speed (1.5-Mbps) operations. Low speed is only supported in host mode.
  - Host mode direct connect of full- and low-speed devices
  - Supports USB on-the-go mode when using an external ULPI PHY, which includes both device and host functionality
  - Host and device support
- Real time clock (RTC) module
  - 32-bit RTC counter, which:
    - Increments for every one second
    - Can be initialized by software to a specific initial count value
  - An alarm function with programmable and maskable alarm interrupt
  - Programmable and maskable every second interrupt
  - Two possible clock sources:
    - External RTC clock (RTC\_PIT\_CLK)
    - CSB bus clock
  - RTC function can be disabled, if required

## 1.2 MPC8308 Architecture Overview

The following sections describe the major functional units of this device.

### 1.2.1 e300 Core

The device contains the e300c3 processor core, which is an enhanced version of the MPC603e core (used in previous generations of PowerQUICC II processors). Enhancements include integrated parity checking, dual integer units, and other performance-enhancing features. The e300 core is upward software-compatible with existing MPC603e core-based products.

For detailed information regarding the processor core refer to the following:

- The *e300 Power Architecture™ Core Family Reference Manual* (chapters describing the programming model, cache model, memory management model, exception model, and instruction timing) (Document No. E300CORERM)
- The *Programming Environments Manual for 32-Bit Implementations of the PowerPC™ Architecture* (Document No. MPCFPE32B)

The e300 core is a low-power implementation of the family of microprocessors that implements Power Architecture technology. The core implements the 32-bit portion of the architecture that provides 32-bit effective addresses, integer data types of 8, 16, and 32 bits, and floating-point data types of 32 and 64 bits.

The core is a superscalar processor that can issue three instructions (two plus a branch) and completes and retires as many as two instructions per clock cycle. Instructions can execute out of order for increased performance; however, the core makes completion appear sequential.

The e300c3 core integrates six execution units—two integer units (IU1 and IU2) with full multiply and divides, a floating-point unit (FPU), a branch processing unit (BPU) with static branch prediction, a load/store unit (LSU) for data transfers, a performance monitor, and a system register unit (SRU). The ability to execute five instructions in parallel and the use of simple instructions with rapid execution times yield high efficiency and throughput. Most integer instructions execute in one clock cycle; two integer instructions may be executed at the same time with the dual integer units. The FPU is pipelined so a single-precision multiply-add instruction can be issued and completed every clock cycle.

The e300c3 core provides independent on-chip, 16-Kbyte, eight-way set-associative, physically addressed instruction and data caches with parity and integrated way lock capabilities. The processor also features independent on-chip instruction and data memory management units (MMUs). The MMUs contain 64-entry, two-way set-associative, data and instruction translation lookaside buffers (DTLB and ITLB) that provide support for demand-paged virtual memory address translation. The caches use a pseudo least recently used (PLRU) replacement algorithm; the TLBs use a least recently used (LRU) replacement algorithm. The processor also supports block address translation through the use of two independent instruction and data block address translation (IBAT and DBAT) arrays of eight entries each. Effective addresses are compared simultaneously with all eight entries in the BAT array during block translation. In accordance with the architecture, if an effective address hits in both the TLB and BAT array, the BAT translation takes priority.

As an added feature to the e300 core, the device can lock the contents of three of the four ways in the instruction and data cache (or an entire cache). For example, this allows embedded applications to lock

interrupt routines or other important (time-sensitive) instruction sequences into the instruction cache. It allows data to be locked into the data cache, which may be important to code that must have deterministic execution.

The e300 core has high-performance 64-bit data bus and 32-bit address bus interfaces to the rest of the device. The e300 core supports single-beat and burst data transfers for memory accesses and memory-mapped I/O operations.



Figure 1-2 provides a block diagram of the e300 core that shows how the execution units (IU1, IU2, FPU, BPU, LSU, and SRU) operate independently and in parallel. Note that this is a conceptual diagram that does not attempt to show how these features are physically implemented on the chip.

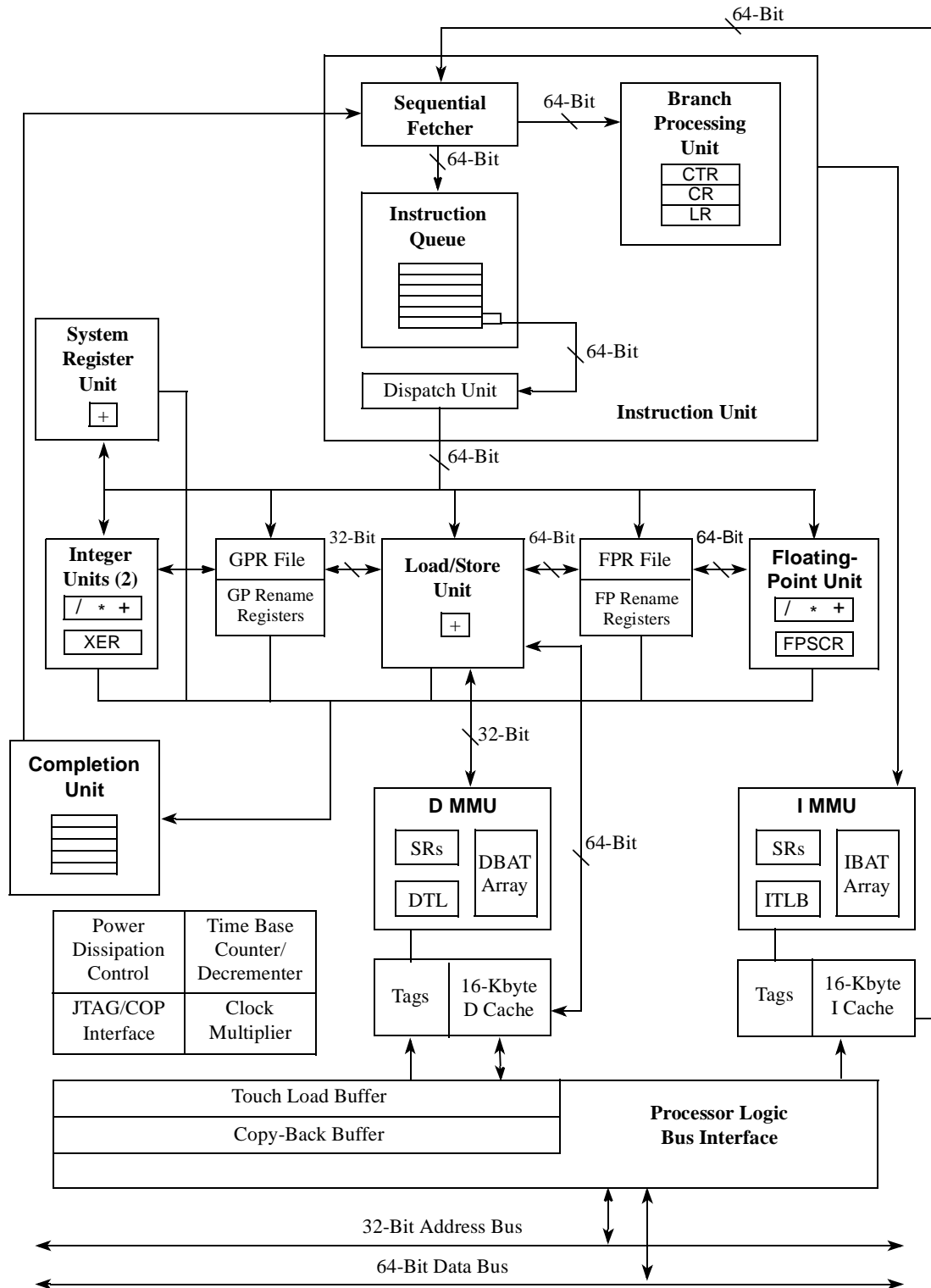


Figure 1-2. MPC8308 Integrated e300c3 Core Block Diagram

## 1.2.2 DDR2 Memory Controller

This fully programmable DDR2 SDRAM controller supports most JEDEC standard  $\times 8$  or  $\times 16$  DDR2 memories available today, including buffered and unbuffered DIMMs. However, mixing non registered and registered DIMMs in the same system is not supported.

The DDR memory controller includes the following features:

- Support for DDR2 SDRAM
- 16- or 32-bit SDRAM data bus
- Programmable settings for meeting all SDRAM timing parameters
- Many different SDRAM configurations supported
  - Support for two physical banks (chip selects)
  - Support for 64-Mbit to 1-Gbit devices with  $\times 8/\times 16$  data ports. Some 2-Gbit devices are supported depending on the internal device configuration.
  - Support for unbuffered and registered DIMMs
- Support for data mask signals and read-modify-write operations for sub-double word writes
- Four-entry input request queue
- Open page management (dedicated entry for each sub-bank)

## 1.2.3 Dual Enhanced Three-Speed Ethernet Controllers

The MPC8308 has two on-chip enhanced three-speed Ethernet controllers. The eTSECs incorporate a media access control (MAC) sublayer that supports 10- and 100-Mbps and 1-Gbps Ethernet/IEEE Std. 802.3 networks with MII and RGMII physical interfaces. The eTSECs include 2-Kbyte receive and 10-Kbyte transmit FIFOs and DMA functions. They also support IEEE Std. 1588.

The buffer descriptors are based on the MPC8260 and MPC860T 10/100 Ethernet programming models. Each eTSEC can emulate a PowerQUICC III TSEC, allowing existing driver software to be re-used with minimal change.

The MPC8308 eTSECs support programmable CRC generation and checking, RMON statistics, and jumbo frames of up to 9.6 Kbytes.

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.

TCP/IP acceleration can be performed at a number of levels. The eTSEC can parse frames at layer 2 of the stack only (Ethernet headers and switching headers), layers 2 to 3 (including IP v4 or IP v6), or layers 2 to 4 (including TCP and UDP).

On receive, the eTSEC provides protocol header recognition, header verification (IP v4 header checksum verification), and TCP/UDP payload checksum verification including verification of associated pseudo-header checksums. On transmit, the eTSEC provides IP v4 and TCP/UDP header checksum generation. The eTSEC does not checksum transmitted packets with IP header options or IP fragments.

To provide for quality of service, transmission from up to eight queues is supported with priority-based queue selection. Arbitration is a modified weighted round-robin queue selection with fair bandwidth allocation.

On receive, packets may be distributed to any of the 64 virtual receive queues overlaid onto the 8 physical receive queues. A table-oriented queue filing strategy is provided based on 16 header fields or flags. Frame rejection is supported for filtering applications.

Filing can be 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, or user-defined bit fields.

## 1.2.4 SerDes PHY

The SerDes PHY block includes the SerDes PHY, the protocol converter per protocol, the protocol mux, and the control registers and control logic.

The SerDes PHY block has the following features:

- Support for one  $\times 1$  PCI Express interface
- Link-layer interfaces to PCI Express
- Memory-mapped registers with 256-byte address region
- SerDes power-down/reset state machine for cold (power-on) or warm (software-initiated) reset of SerDes, PHY, and controllers

The SerDes PHY block supports the following mode of operation:

- One lane running  $\times 1$  PCI Express at 2.5 Gbps

## 1.2.5 PCI Express Interface

The MPC8308 supports a PCI Express interface compliant with the *PCI Express Base Specification Revision 1.0a*. It is able to act as either root complex or endpoint. It only supports virtual channel 0 (VC0) and eight traffic classes (TC0–TC7). The maximum supported packet payload size is 128 bytes.

The physical layer supports single  $\times 1$  lane width running at the specified data rate of 2.5 Gbauds.

Inbound  $INTx$  transaction is supported and change the state of a level-sensitive interrupt presented to the PIC. Outbound  $INTx$  transaction is supported. Message signaled interrupt (MSI) transaction is supported and controls up to 256 interrupt sources within the PIC. Outbound MSI transaction may be created by software using the MSI Capability Register Sets.

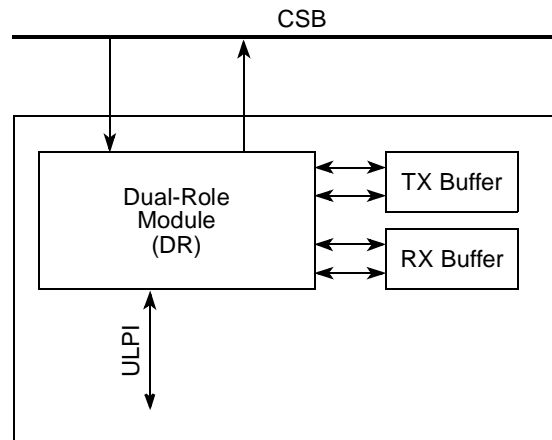
The physical layer of the PCI Express interface operates at a 2.5-Gbaud data rate.

## 1.2.6 Universal Serial Bus (USB) 2.0

The USB 2.0 controller offers operation as a host or device. The USB controller provides point-to-point connectivity, which complies with the Universal Serial Bus Revision 2.0 Specification. The USB controllers can be configured to operate as a stand-alone host or stand-alone device. See [Figure 1-3](#) for more information.

The host and device functions are both configured to support the following four types of USB transfers:

- Bulk
- Control
- Interrupt
- Isochronous



**Figure 1-3. USB Controllers Port Configuration**

### 1.2.6.1 USB Dual-Role Controller

- Designed to comply with *Universal Serial Bus Revision 2.0 Specification*
- Supports operation as a stand-alone USB host controller
  - Supports USB root hub with one downstream-facing port
  - Enhanced host controller interface (EHCI) compatible
- Supports operation as a stand-alone USB device
  - Supports one upstream-facing port
  - Supports three programmable bi-directional USB endpoints
- Supports high-speed (480-Mbps), full-speed (12-Mbps), and low-speed (1.5-Mbps) operations. Low speed is only supported in host mode.
- Host mode direct connect of full-speed and low-speed devices
- Supports USB on-the-go mode when using an external ULPI PHY that includes both device and host functionality
- Host and device support

### 1.2.7 Enhanced Local Bus Controller (eLBC)

The main component of the enhanced local bus controller (eLBC) is its memory controller that provides a seamless interface to many types of memory devices and peripherals. The memory controller is responsible for controlling four memory banks shared by a NAND Flash control machine (FCM), a general-purpose chip-select machine (GPCM), and up to three user-programmable machines (UPMs). As

such, it supports a minimal glue logic interface to SRAM, EPROM, NOR Flash EPROM, NAND Flash EPROM, Flash EPROM, burstable RAM, and other peripherals.

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.

The eLBC provides two Write Enable signals to allow single-byte write access to external 16-bit eLBC slave devices.

The main features of the enhanced local bus controller (eLBC) are as follows:

- Memory controller with four memory banks (chip selects)
  - 32-bit address decoding with mask
  - Variable memory block sizes (32 Kbytes to 2 Gbytes in FCM mode, 32 Kbytes to 64 Mbytes in UPM mode, and 32 Kbytes to 64 Mbytes in GPCM mode)
  - Selection of control signal generation on a per-bank basis
  - Data buffer controls activated on a per-bank basis
  - Up to 256-byte bursts, arbitrarily aligned
  - Automatic segmentation of large transactions into memory accesses optimized for bus width and addressing capability
  - Write-protection capability
  - Atomic operation
- General-purpose chip-select machine (GPCM)
  - Compatible with SRAM, EPROM, NOR Flash EEPROM, FEPRM, and peripherals
  - Global (boot) chip-select available at system reset
  - Boot chip-select support for 8- and 16-bit devices
  - Minimum three-clock access to external devices
  - Two byte-write-enable signals ( $\overline{\text{LWE}}[0:1]$ )
  - 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
  - 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 devices with depths of 64, 128, 256, and 512 Kbytes, and 1, 2, 4, 8, 16, 32, 64 Mbytes
  - Support for 8- and 16-bit devices
  - Page mode support for successive transfers within a burst
- Optional monitoring of transfers between local bus internal masters and local bus slaves (local bus error reporting)

### 1.2.8 Integrated Programmable Interrupt Controller (IPIC)

The Integrated Programmable Interrupt Controller (IPIC) implements the necessary functions to provide a flexible solution for general-purpose interrupt control. The IPIC includes the following features:

- Functional and programming models are compatible with the MPC8260 interrupt controller
- Support for external and internal discrete interrupt sources
- Support for one external (optional) and seven internal machine checkstop interrupt sources
- Programmable highest priority request
- Two programmable priority mixed groups of four on-chip and four external interrupt signals with two priority schemes for each group: grouped and spread
- Four programmable priority internal groups of eight on-chip interrupt signals with two priority schemes for each group: grouped and spread
- Priority interrupts can be programmed to support a critical ( $\overline{cint}$ ) or system management ( $\overline{smi}$ ) interrupt type
- External and internal interrupts directed to a host processor
- Unique vector number for each interrupt source
- IPIC can support external interrupt request with programmable triggering mechanism. It can be programmed to use one of the following mechanisms:
  - Active low level triggering
  - Active high level triggering
  - Raising edge triggering
  - Falling edge triggering

## 1.2.9 I<sup>2</sup>C Interface

The inter-IC (IIC or I<sup>2</sup>C) 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 the system and other devices, such as microcontrollers, EEPROMs, real-time clock devices, A/D converters, and LCDs. The two-wire bus minimizes the interconnections between devices. The synchronous, multi-master bus of the I<sup>2</sup>C allows the connection of additional devices to the bus for expansion and system development.

The I<sup>2</sup>C controller is a true multi-master bus, which includes collision detection and arbitration that prevents data corruption if two or more masters attempt to control the bus simultaneously. This feature allows for complex applications with multiprocessor control. The I<sup>2</sup>C controller consists of a transmitter/receiver unit, clocking unit, and control unit. The I<sup>2</sup>C unit supports general broadcast mode and on-chip filtering rejects spikes on the bus.

The I<sup>2</sup>C interface includes the following features:

- Two-wire interface
- Multi-master operational
- Arbitration lost interrupt with automatic mode switching from master to slave
- Calling address identification interrupt
- Bus busy detection
- Software-programmable clock frequency
- Software-selectable acknowledge bit
- On-chip filtering for spikes on the bus
- Address broadcasting supported

## 1.2.10 General Purpose DMA Controller

The direct memory access (DMA) is capable of performing complex data transfers with minimal intervention from a host processor via two programmable channels. The hardware architecture includes a DMA engine, which performs source and destination address calculations, and the actual data movement operations, along with a local memory containing the transfer control descriptors (TCD) for the channels. This SRAM-based implementation is utilized to minimize the overall module size.

The DMA is a highly programmable data transfer engine that has been optimized to minimize the required intervention from the host processor. It is intended for use in applications where the data size to be transferred is statically known, and is not defined within the data packet itself. The DMA hardware supports:

- Single design with two channels (Tx and Rx)
- 32-byte transfer control descriptor per channel stored in local memory
- 32 bytes of data registers, used as temporary storage to support burst transfers

### 1.2.11 Dual Universal Asynchronous Receiver/Transmitter (DUART)

The device includes a Dual universal Asynchronous Receiver/Transmitter (DUART) intended for use in maintenance, bring up, and debug systems. The device provides a standard two-wire data (TXD and RXD) for each port. The DUART is a slave interface. An interrupt is provided to the interrupt controller. Interrupts are generated for transmit, receive, and line status.

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

Software programmable baud rate generators divide the system clock to generate a 16x clock. Serial interface data formats (data length, parity, 1/1.5/2 STOP bit, baud rate) are also software selectable.

The DUART includes the following features:

- Full-duplex operation
- Programming model compatible with the original PC16450 UART and the PC16550D (an improved version of the 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, and line status interrupts
- Software-programmable baud rate generators that divide the system clock by 1 to  $(2^{16} - 1)$  and generate a 16x clock for the transmitter and receiver engines
- Software-selectable serial-interface data format (data length, parity, 1/1.5/2 STOP bit, baud rate)
- Line status registers
- Line-break detection and generation
- Internal diagnostic support, local loopback, and break functions
- Prioritized interrupt reporting
- Overrun, parity, and framing error detection

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

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.

- 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.

- SDIO

Under the SD protocol, the SD cards 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. The I/O card provides high-speed data I/O with low power consumption for mobile electronic devices. The combo card has both memory and I/O functions.

The eSDHC acts as a bridge, passing host bus transactions to SD/SDIO/MMC cards by sending commands and performing data accesses to or from the cards. It handles the SD/SDIO/MMC protocol at the transmission level.

The eSDHC can select the following modes for data transfer:

- SD 1-bit
- SD 4-bit
- MMC 1-bit
- MMC 4-bit
- Identification mode (up to 400 KHz)
- Full-speed mode (up to 25 MHz) or high-speed mode (up to 50 MHz)

### 1.2.13 System Timers

The system includes the following timers:

- Periodic interrupt timer
- Real time clock
- Software watchdog timer
- One general-purpose timer block, supporting four 16-bit programmable timers or two cascaded 32-bit timers, or one cascaded 64-bit counter



# Chapter 2

## Signal Descriptions

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

- Overview of signals and cross references for signals that serve multiple functions, including a list ordered by functional block and a list by alphabetical order.
- List of output signal states at reset

### NOTE

A bar over a signal name indicates that the signal is active low, such as  $\overline{\text{MWE}}$ . 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 TSEC1\_RX\_DV (interrupt input), are referred to as asserted when they are high and negated when they are low.

## 2.1 Signals Overview

The signals are grouped as follows:

- DDR2 memory interface signals
- DUART interface signals
- I<sup>2</sup>C interface signals
- Ethernet management interface signals
- eTSEC1 and eTSEC2 interface signals
- PCI Express PHY signals
- Enhanced local bus interface signals
- GPIO interface signals
- Global timers/USB interface signals
- IPIC interface signals
- SPI interface signals
- JTAG interface signals
- System control signals
- Test interface signals
- Clock interface signals
- eSDHC interface signals
- Miscellaneous signals
- IEEE 1588 signals

Figure 2-1 and Figure 2-2 show the external signals of the device and how the signals are grouped. Refer to the *PowerQUICC II Pro MPC8308 Hardware Specification* for a pinout diagram showing pin numbers and a listing of all the electrical and mechanical specifications.

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

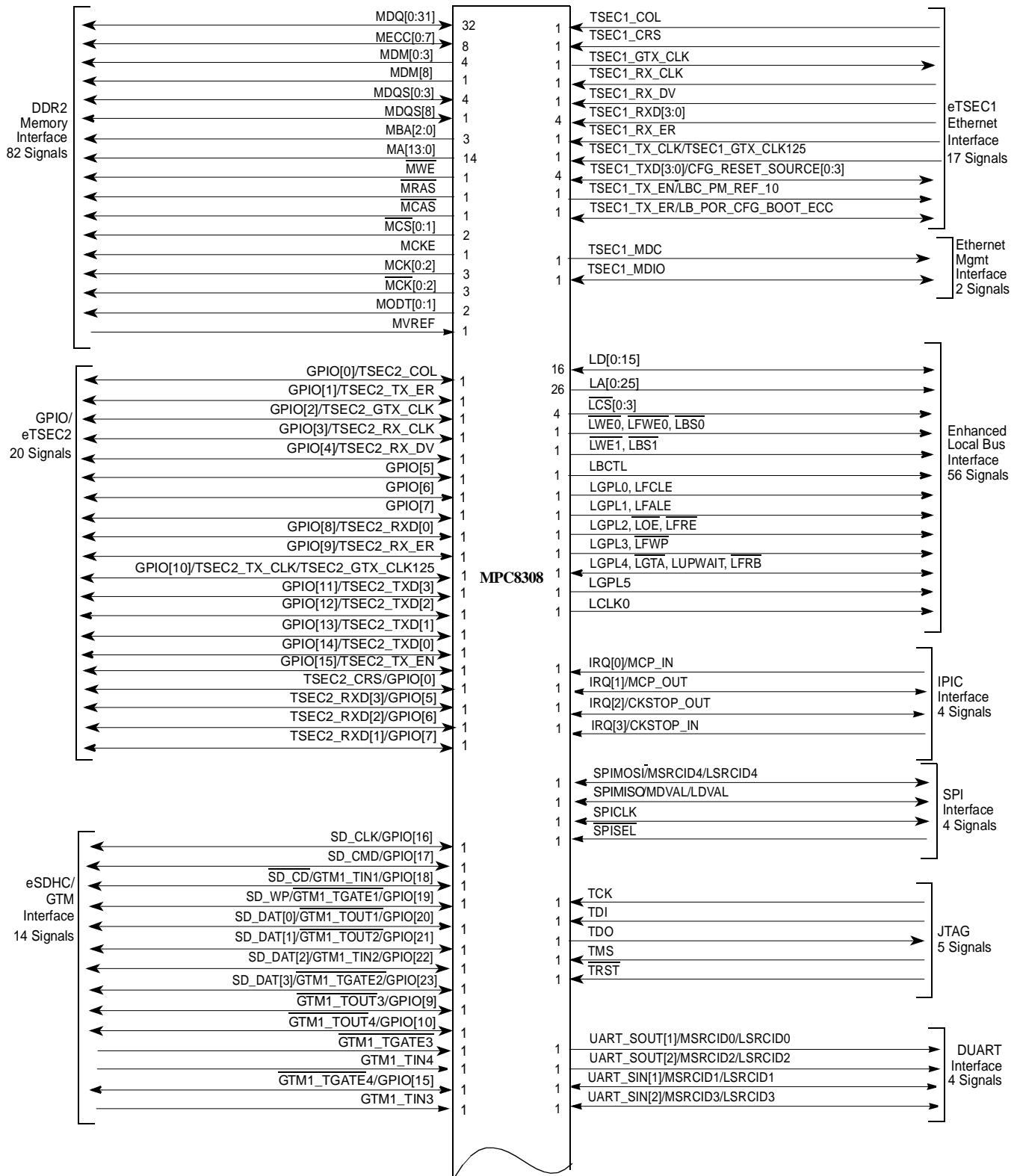


Figure 2-1. MPC8308 Signal Groupings (1 of 2)

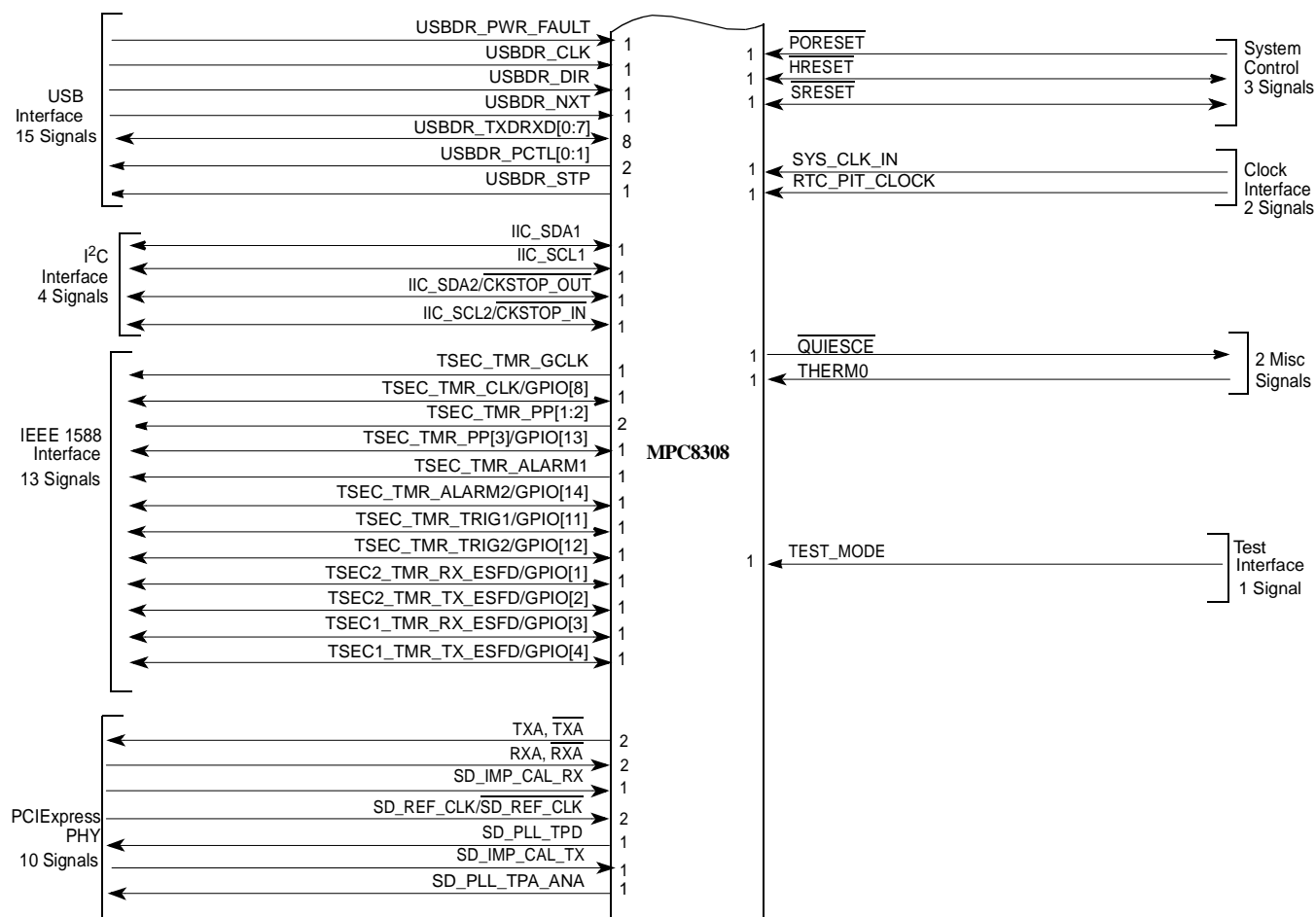


Figure 2-2. MPC8308 Signal Groupings (2 of 2)

The following tables provide summaries of signal functions. [Table 2-1](#) provides a summary of the signals grouped by function and [Table 2-2](#) a summary of the signals grouped alphabetically. These tables detail the signal name, interface, alternate functions, number of signals, and whether the signal is an input, output, or bidirectional. Finally, the table provides a pointer to another table where the signal function is described.

Table 2-1. MPC8308 Signal Reference by Functional Block

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
MDQ[0:31]	DDR data	DDR2	32	I/O	<a href="#">9-1/9-4</a>	—	—
MECC[0:7]	DDR ECC data	DDR2	8	I/O	<a href="#">9-1/9-4</a>	—	—
MDM[0:3]	DDR data mask	DDR2	4	O	<a href="#">9-1/9-4</a>	—	—
MDM[8]	DDR data mask	DDR2	1	O	<a href="#">9-1/9-4</a>	—	—
MDQS[0:3]	DDR data strobe	DDR2	4	I/O	<a href="#">9-1/9-4</a>	—	—
MDQS[8]	DDR data strobe	DDR2	1	I/O	<a href="#">9-1/9-4</a>	—	—

Table 2-1. MPC8308 Signal Reference by Functional Block (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
MBA[2:0]	DDR bank select	DDR2	3	O	9-1/9-4	—	—
MA[13:0]	DDR address	DDR2	14	O	9-1/9-4	—	—
$\overline{\text{MWE}}$	DDR write enable	DDR2	1	O	9-1/9-4	—	—
$\overline{\text{MRAS}}$	DDR row address strobe	DDR2	1	O	9-1/9-4	—	—
$\overline{\text{MCAS}}$	DDR column address strobe	DDR2	1	O	9-1/9-4	—	—
$\overline{\text{MCS}}[0:1]$	DDR chip select (2/DIMM)	DDR2	2	O	9-1/9-4	—	—
MCKE	DDR clock enable	DDR2	1	O	9-1/9-4	—	—
MCK[0:2]	DDR differential clocks	DDR2	3	O	9-1/9-4	—	—
$\overline{\text{MCK}}[0:2]$	DDR differential clocks	DDR2	3	O	9-1/9-4	—	—
MODT[0:1]	DRAM on-die termination	DDR2	2	O	9-1/9-4	—	—
MVREF	DDR2 DRAM reference	DDR2	1	PWR	9-1/9-4	—	—
TSEC1_COL	eTSEC1 collision detect	eTSEC1	1	I	16-2/16-7	—	—
TSEC1_CRS	eTSEC1 carrier sense	eTSEC1	1	I	16-2/16-7	—	—
TSEC1_GTX_CLK	eTSEC1 transmit clock out	eTSEC1	1	O	16-2/16-7	—	—
TSEC1_RX_CLK	eTSEC1 receive clock	eTSEC1	1	I	16-2/16-7	—	—
TSEC1_RX_DV	eTSEC1 receive data valid	eTSEC1	1	I	16-2/16-7	—	—
TSEC1_RXD[3:0]	eTSEC1 receive data 3–0	eTSEC1	4	I	16-2/16-7	—	—
TSEC1_RX_ER	eTSEC1 receiver error	eTSEC1	1	I	16-2/16-7	—	—
TSEC1_TX_CLK	eTSEC1 transmit clock in	eTSEC1	1	I	16-2/16-7	TSEC1_GTX_CLK125	16-2/16-7
TSEC1_TXD[3:0]	eTSEC1 transmit data 3–0	eTSEC1	4	I/O	16-2/16-7	CFG_RESET_SOURCE[0:3]	4-1/4-1
TSEC1_TX_EN	eTSEC1 transmit enable	eTSEC1	1	O	16-2/16-7	LBC_PM_REF_10	—

Table 2-1. MPC8308 Signal Reference by Functional Block (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
TSEC1_TX_ER	eTSEC1 transmit error	eTSEC1	1	I/O	16-2/16-7	LB_POR_CFG_BOOT_ECC	—
TSEC1_GTX_CLK125	Gigabit reference clock	eTSEC1	1	I	16-2/16-7	TSEC1_TX_CLK	16-2/16-7
TSEC2_COL	eTSEC2 collision detect	eTSEC2	1	I/O	16-2/16-7	GPIO[0]	21-1/21-2
TSEC2_TX_ER	eTSEC2 transmit error	eTSEC2	1	I/O	16-2/16-7	GPIO[1]	21-1/21-2
TSEC2_GTX_CLK	eTSEC2 transmit clock out	eTSEC2	1	I/O	16-2/16-7	GPIO[2]	21-1/21-2
TSEC2_RX_CLK	eTSEC2 receive clock	eTSEC2	1	I/O	16-2/16-7	GPIO[3]	21-1/21-2
TSEC2_RX_DV	eTSEC2 receive data valid	eTSEC2	1	I/O	16-2/16-7	GPIO[4]	21-1/21-2
TSEC2_RXD[3:1]	eTSEC2 receive data 3–1	eTSEC2	3	I/O	16-2/16-7	GPIO[5:7]	21-1/21-2
TSEC2_RXD[0]	eTSEC2 receive data 0	eTSEC2	1	I/O	16-2/16-7	GPIO[8]	21-1/21-2
TSEC2_RX_ER	eTSEC2 receiver error	eTSEC2	1	I/O	16-2/16-7	GPIO[9]	21-1/21-2
TSEC2_TX_CLK	eTSEC2 transmit clock in	eTSEC2	1	I/O	16-2/16-7	GPIO[10]/ TSEC2_GTX_CLK125	21-1/21-2/ 21-1/21-2
TSEC2_GTX_CLK125	Gigabit reference clock	eTSEC2	1	I/O	16-2/16-7	GPIO[10]/ TSEC2_TX_CLK	21-1/21-2/ 21-1/21-2
TSEC2_TXD[3:0]	eTSEC2 transmit data 3–0	eTSEC2	4	I/O	16-2/16-7	GPIO[11:14]	21-1/21-2
TSEC2_TX_EN	eTSEC2 transmit enable	eTSEC2	1	I/O	16-2/16-7	GPIO[15]	21-1/21-2
TSEC2_CRS	eTSEC2 carrier sense	eTSEC2	1	I/O	16-2/16-7	GPIO[0]	21-1/21-2
GPIO[0]	General-purpose I/O signal	GPIO	1	I/O	21-1/21-2	TSEC2_COL	16-2/16-7
GPIO[1]	General-purpose I/O signal	GPIO	1	I/O	21-1/21-2	TSEC2_TX_ER	16-2/16-7
GPIO[2]	General-purpose I/O signal	GPIO	1	I/O	21-1/21-2	TSEC2_GTX_CLK	16-2/16-7
GPIO[3]	General-purpose I/O signal	GPIO	1	I/O	21-1/21-2	TSEC2_RX_CLK	16-2/16-7
GPIO[4]	General-purpose I/O signal	GPIO	1	I/O	21-1/21-2	TSEC2_RX_DV	16-2/16-7

Table 2-1. MPC8308 Signal Reference by Functional Block (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
GPIO[5:7]	General-purpose I/O signal	GPIO	3	I/O	21-1/21-2	—	—
GPIO[8]	General-purpose I/O signal	GPIO	1	I/O	21-1/21-2	TSEC2_RXD[0]	16-2/16-7
GPIO[9]	General-purpose I/O signal	GPIO	1	I/O	21-1/21-2	TSEC2_RX_ER	16-2/16-7
GPIO[10]	General-purpose I/O signal	GPIO	1	I/O	21-1/21-2	TSEC2_TX_CLK/ TSEC2_GTX_CLK125	16-2/16-7
GPIO[11:14]	General-purpose I/O signal	GPIO	4	I/O	21-1/21-2	TSEC2_TXD[3:0]	16-2/16-7
GPIO[15]	General-purpose I/O signal	GPIO	1	I/O	21-1/21-2	TSEC2_TX_EN	16-2/16-7
TSEC_TMR_GCLK	1588 clock-out	eTSEC	1	O	16-2/16-7	—	—
TSEC_TMR_CLK	1588 clock-in	eTSEC	1	I/O	16-2/16-7	GPIO[8]	21-1/21-2
TSEC_TMR_PP[1:2]	1588 timer pulse-out 1, 2	eTSEC	2	O	16-2/16-7	—	—
TSEC_TMR_PP[3]	1588 timer pulse-out 3	eTSEC	1	I/O	16-2/16-7	GPIO[13]	21-1/21-2
TSEC_TMR_ALARM[1]	1588 timer alarm-out 1	eTSEC	1	O	16-2/16-7	—	—
TSEC_TMR_ALARM[2]	1588 timer alarm-out 2	eTSEC	1	I/O	16-2/16-7	GPIO[14]	21-1/21-2
TSEC_TMR_TRIG[1:2]	1588 trigger-in 1, 2	eTSEC	2	I/O	16-2/16-7	GPIO[11:12]	21-1/21-2
TSEC2_TMR_RX_ESFD	Receive external start of frame delimiter (for eTSEC2)	eTSEC	1	I/O	16-2/16-7	GPIO[1]	21-1/21-2
TSEC2_TMR_TX_ESFD	Transmit external start of frame delimiter (for eTSEC2)	eTSEC	1	I/O	16-2/16-7	GPIO[2]	21-1/21-2
TSEC1_TMR_RX_ESFD	Transmit external start of frame delimiter (for eTSEC1)	eTSEC	1	I/O	16-2/16-7	GPIO[3]	21-1/21-2
TSEC1_TMR_TX_ESFD	Receive external start of frame delimiter (for eTSEC1)	eTSEC	1	I/O	16-2/16-7	GPIO[4]	21-1/21-2
USBDR_PWR_FAULT	USB VBus power fault	USB	1	I	13-2/13-3	—	—



Table 2-1. MPC8308 Signal Reference by Functional Block (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
USBDR_CLK	Clocking signal for ULPI PHY interface	USB	1	I	13-2/13-3	—	—
USBDR_DIR	Direction of data bus	USB	1	I	13-2/13-3	—	—
USBDR_NXT	Nest data	USB	1	I	13-2/13-3	—	—
USBDR_TXDRXD[0:7]	Data bit 0–7	USB	8	I/O	13-2/13-3	—	—
USBDR_PCTL[0:1]	Port control 0–1	USB	2	O	13-2/13-3	—	—
USBDR_STP	End of a transfer on the bus	USB	1	O	13-2/13-3	—	—
TSEC1_MDC	Ethernet management data clock	Ethernet Management	1	O	16-2/16-7	—	—
TSEC1_MDIO	Ethernet management data in/out	Ethernet Management	1	I/O	16-2/16-7	—	—
CFG_RESET_SOURCE[0:3]	Reset configuration word source selection	Reset and clock	4	I	4-1/4-1	TSEC1_TXD[3:0]	16-2/16-7
$\overline{\text{CKSTOP\_IN}}$	Checkstop in	Reset and clock	1	I/O	—	IIC_SCL2/ $\overline{\text{IRQ3}}$	17-1/17-3, 8-1/8-5
$\overline{\text{CKSTOP\_OUT}}$	Checkstop out	Reset and clock	1	I/O	—	IIC_SDA2/ $\overline{\text{IRQ2}}$	17-1/17-3, 8-1/8-5
TXA	Serial transmitter, lane A, positive data	PCI Express PHY	1	O	15-1/15-2	—	—
$\overline{\text{TXA}}$	Serial transmitter, lane A, negative data (complement)	PCI Express PHY	1	O	15-1/15-2	—	—
RXA	Serial receiver, lane A, positive data	PCI Express PHY	1	I	15-1/15-2	—	—
$\overline{\text{RXA}}$	Serial receiver, lane A, negative data (complement)	PCI Express PHY	1	I	15-1/15-2	—	—
SD_IMP_CAL_RX	Receiver impedance control signal	PCI Express PHY	1	I	15-1/15-2	—	—
SD_REF_CLK	SerDes PLL reference clock	PCI Express PHY	1	I	15-1/15-2	—	—

Table 2-1. MPC8308 Signal Reference by Functional Block (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
$\overline{\text{SD\_REF\_CLK}}$	SerDes PLL reference clock (complement)	PCI Express PHY	1	I	<a href="#">15-1/15-2</a>	—	—
SD_PLL_TPD	Digital test point for SerDes PLL testing	PCI Express PHY	1	O	—	—	—
SD_IMP_CAL_TX	Transmitter impedance control signal	PCI Express PHY	1	I	<a href="#">15-1/15-2</a>	—	—
XCOREVDD[0:2]	SerDes transceiver core supply	PCI Express PHY	3	PWR	—	—	—
XCOREVSS[0:2]	SerDes transceiver core ground	PCI Express PHY	3	GND	—	—	—
XPADVDD[0:1]	SerDes transceiver pad supply	PCI Express PHY	2	PWR	—	—	—
XPADVSS[0:1]	SerDes transceiver pad ground	PCI Express PHY	2	GND	—	—	—
SDAVDD	Analog supply for SerDes PLL	PCI Express PHY	1	PWR	—	—	—
SD_PLL_TPA_ANA	Analog test point for SerDes PLL testing	PCI Express PHY	1	O	—	—	—
SDAVSS	Analog ground for SerDes PLL	PCI Express PHY	1	GND	—	—	—
LB_POR_CFG_BOOT_ECC	Enable/Disable ECC for Flash during RCW load	eLBC	1	I	—	TSEC1_TX_ER	<a href="#">16-2/16-7</a>
LBC_PM_REF_10	Status of ECC error during boot loading from flash	eLBC	1	O	—	TSEC1_TX_EN	<a href="#">16-2/16-7</a>
LD[0:15]	LBC data	eLBC	16	I/O	<a href="#">10-2/10-5</a>	—	—
LA[0:25]	LBC port address	eLBC	26	O	<a href="#">10-2/10-5</a>	—	—
$\overline{\text{LCS}}[0:3]$	LBC chip select 0–3	eLBC	4	O	<a href="#">10-2/10-5</a>	—	—
$\overline{\text{LWE0}}, \overline{\text{LWE0}}, \overline{\text{LBS0}}$	LBC write enable, Byte (lane) select	eLBC	1	O	<a href="#">10-2/10-5</a>	—	—
$\overline{\text{LWE1}}, \overline{\text{LBS1}}$	LBC write enable, Byte (lane) select	eLBC	1	O	<a href="#">10-2/10-5</a>	—	—
LBCTL	LBC data buffer control	eLBC	1	O	<a href="#">10-2/10-5</a>	—	—

Table 2-1. MPC8308 Signal Reference by Functional Block (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
LGPL0, LFCLE	LBC UPM general purpose line 0, Flash command latch enable	eLBC	1	O	10-2/10-5	—	—
LGPL1, LFALE	LBC GP line 1, Flash address latch enable	eLBC	1	O	10-2/10-5	—	—
LGPL2, $\overline{\text{LOE}}$ , $\overline{\text{LFRE}}$	LBC GP line 2, LBC output enable, Flash read enable	eLBC	1	O	10-2/10-5	—	—
LGPL3, $\overline{\text{LFWP}}$	LBC GP line 3, Flash write protect	eLBC	1	O	10-2/10-5	—	—
LGPL4, $\overline{\text{LGTA}}$ , LUPWAIT, $\overline{\text{LFRB}}$	LBC GP line 4, Transaction termination, External device wait, Flash ready/busy	eLBC	1	I/O	10-2/10-5	—	—
LGPL5	LBC GP line 5	eLBC	1	O	10-2/10-5	—	—
LCLK0	LBC clock	eLBC	1	O	10-2/10-5	—	—
IIC_SDA1	I <sup>2</sup> C serial data 1	I <sup>2</sup> C	1	I/O	17-1/17-3	—	—
IIC_SDA2	I <sup>2</sup> C serial data 2	I <sup>2</sup> C	1	I/O	17-1/17-3	$\overline{\text{CKSTOP\_OUT}}$	—
IIC_SCL1	I <sup>2</sup> C serial clock 1	I <sup>2</sup> C	1	I/O	17-1/17-3	—	—
IIC_SCL2	I <sup>2</sup> C serial clock 2	I <sup>2</sup> C	1	I/O	17-1/17-3	$\overline{\text{CKSTOP\_IN}}$	—
UART_SOUT[1]	DUART serial data out	DUART	1	O	18-1/18-3	MSRCID0/LSRCID0	—, 10-2/10-5
UART_SOUT[2]	DUART serial data out	DUART	1	O	18-1/18-3	MSRCID2/LSRCID2	—, 10-2/10-5
UART_SIN[1]	DUART serial data in	DUART	1	I/O	18-1/18-3	MSRCID1/LSRCID1	—, 10-2/10-5
UART_SIN[2]	DUART serial data in	DUART	1	I/O	18-1/18-3	MSRCID3/LSRCID3	—, 10-2/10-5
SPIMOSI	SPI master-out slave-in	SPI	1	I/O	19-1/19-6	MSRCID4/LSRCID4	—, 10-2/10-5
SPIMISO	SPI master-in slave-out	SPI	1	I/O	19-1/19-6	MDVAL/LDVAL	—, 10-2/10-5
SPICLK	SPI clock	SPI	1	I/O	19-1/19-6	—	—
$\overline{\text{SPISEL}}$	SPI slave select	SPI	1	I	19-1/19-6	—	—

Table 2-1. MPC8308 Signal Reference by Functional Block (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
$\overline{\text{IRQ}}[0]/\overline{\text{MCP\_IN}}$	External interrupt 0	IPIC	1	I	8-1/8-5	—	—
$\overline{\text{IRQ}}[1]$	External interrupt 1	IPIC	1	I/O	8-1/8-5	$\overline{\text{MCP\_OUT}}$	8-1/8-5
$\overline{\text{IRQ}}[2]$	External interrupt 2	IPIC	1	I/O	8-1/8-5	$\overline{\text{CKSTOP\_OUT}}$	—
$\overline{\text{IRQ}}[3]$	External interrupt 3	IPIC	1	I	8-1/8-5	$\overline{\text{CKSTOP\_IN}}$	—
$\overline{\text{MCP\_OUT}}$	Machine check interrupt output	IPIC	1	O	8-1/8-5	$\overline{\text{IRQ}}[1]$	8-1/8-5
TCK	Test clock	JTAG	1	I	20-1/20-2	—	—
TDI	Test data in	JTAG	1	I	20-1/20-2	—	—
TDO	Test data out	JTAG	1	O	20-1/20-2	—	—
TMS	Test mode select	JTAG	1	I	20-1/20-2	—	—
$\overline{\text{TRST}}$	Test reset	JTAG	1	I	20-1/20-2	—	—
TEST_MODE	Internal test signal <b>Note:</b> "This pin must always be tied to VSS"	Test	1	I	—	—	—
$\overline{\text{PORESET}}$	Power on reset	System control	1	I	4-1/4-1	—	—
$\overline{\text{HRESET}}$	Hard reset	System control	1	I/O	4-1/4-1	—	—
$\overline{\text{SRESET}}$	Soft reset	System control	1	I/O	4-1/4-1	—	—
SYS_CLK_IN	Clock input	Clocks	1	I	4-2/4-2	—	—
RTC_PIT_CLOCK	32.768 KHz clock input to RTC	Clocks	1	I	15-1/15-4	—	—
AVDD1	Core PLL power supply	Misc	1	PWR	—	—	—
AVDD2	System PLL power supply	Misc	1	PWR	—	—	—
$\overline{\text{QUIESCE}}$	Quiescent state	Misc	1	O	5-64/5-69	—	—
THERM0	Thermal resistor access 0	Misc	1	I	—	—	—
SD_CLK	eSDHC clock out	eSDHC	1	I/O	11-1/11-4	GPIO[16]	21-1/21-2
SD_CMD	eSDHC command/response signals	eSDHC	1	I/O	11-1/11-4	GPIO[17]	21-1/21-2
$\overline{\text{SD\_CD}}$	Card detection signal	eSDHC	1	I/O	11-1/11-4	GTM1_TIN1/ GPIO[18]	5-53/5-55/ 21-1/21-2

Table 2-1. MPC8308 Signal Reference by Functional Block (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
SD_WP	Write protection signal	eSDHC	1	I/O	11-1/11-4	$\overline{\text{GTM1\_TGATE1}}$ / GPIO[19]	5-53/5-55/ 21-1/21-2
SD_DAT[0]	Data signal 0	eSDHC	1	I/O	11-1/11-4	$\overline{\text{GTM1\_TOUT1}}$ / GPIO[20]	5-53/5-55/ 21-1/21-2
SD_DAT[1]	Data signal 1	eSDHC	1	I/O	11-1/11-4	$\overline{\text{GTM1\_TOUT2}}$ / GPIO[21]	5-53/5-55/ 21-1/21-2
SD_DAT[2]	Data signal 2	eSDHC	1	I/O	11-1/11-4	GTM1_TIN2/ GPIO[22]	5-53/5-55/ 21-1/21-2
SD_DAT[3]	Data signal 3	eSDHC	1	I/O	11-1/11-4	$\overline{\text{GTM1\_TGATE2}}$ / GPIO[23]	5-53/5-55/ 21-1/21-2
GTM1_TIN1	Timer in 1	Global Timers	1	I/O	5-53/5-55	$\overline{\text{SD\_CD}}$ /GPIO[18]	11-1/11-4/ 21-1/21-2
$\overline{\text{GTM1\_TGATE1}}$	Timer gate 1	Global Timers	1	I/O	5-53/5-55	SD_WP/GPIO[19]	11-1/11-4/ 21-1/21-2
$\overline{\text{GTM1\_TOUT1}}$	Timer out 1	Global Timers	1	I/O	5-53/5-55	SD_DAT[0]/GPIO[20]	11-1/11-4/ 21-1/21-2
GTM1_TIN2	Timer in 2	Global Timers	1	I/O	5-53/5-55	SD_DAT[2]/GPIO[22]	11-1/11-4/ 21-1/21-2
$\overline{\text{GTM1\_TGATE2}}$	Timer gate 2	Global Timers	1	I/O	5-53/5-55	SD_DAT[3]/GPIO[23]	11-1/11-4/ 21-1/21-2
$\overline{\text{GTM1\_TOUT2}}$	Timer out 2	Global Timers	1	I/O	5-53/5-55	SD_DAT[1]/GPIO[21]	11-1/11-4/ 21-1/21-2
GTM1_TIN3	Timer in 3	Global Timers	1	I	5-53/5-55	—	—
$\overline{\text{GTM1\_TGATE3}}$	Timer gate 3	Global Timers	1	I	5-53/5-55	—	—
$\overline{\text{GTM1\_TOUT}}[3:4]$	Timer out 3, 4	Global Timers	2	I/O	5-53/5-55	GPIO[9:10]	21-1/21-2
GTM1_TIN4	Timer in 4	Global Timers	1	I	5-53/5-55	—	—
$\overline{\text{GTM1\_TGATE4}}$	Timer gate 4	Global Timers	1	I/O	5-53/5-55	GPIO[15]	21-1/21-2
MSRCID0/LSRCID0	Memory debug source ID	Debug	1	O	—, 10-2/10-5	UART_SOUT[1]	18-1/18-3
MSRCID1/LSRCID1	Memory debug source ID	Debug	1	I/O	—, 10-2/10-5	UART_SIN[1]	18-1/18-3
MSRCID2/LSRCID2	Memory debug source ID	Debug	1	I/O	—, 10-2/10-5	UART_SOUT[2]	18-1/18-3
MSRCID3/LSRCID3	Memory debug source ID	Debug	1	I/O	—, 10-2/10-5	UART_SIN[2]	18-1/18-3
MSRCID4/LSRCID4	Memory debug source ID	Debug	1	O	—, 10-2/10-5	SPIMOSI	19-1/19-6
MDVAL/LDVAL	Memory debug data valid	Debug	1	I/O	—, 10-2/10-5	SPIMISO	19-1/19-6

Table 2-2 lists the signals in alphabetical order.

**Table 2-2. MPC8308 Signal Reference by Alphabetical Order**

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
AVDD1	Core PLL power supply	Misc	1	PWR	—	—	—
AVDD2	System PLL power supply	Misc	1	PWR	—	—	—
CFG_RESET_SOURCE[0:3]	Reset configuration word source selection	Reset and clock	4	I	4-1/4-1	TSEC1_TXD[3:0]	16-2/16-7
$\overline{\text{CKSTOP\_IN}}$	Checkstop in	Reset and clock	1	I/O	—	IIC_SCL2/ $\overline{\text{IRQ3}}$	17-1/17-3, 8-1/8-5
$\overline{\text{CKSTOP\_OUT}}$	Checkstop out	Reset and clock	1	I/O	—	IIC_SDA2/ $\overline{\text{IRQ2}}$	17-1/17-3, 8-1/8-5
GPIO[0]	General-purpose I/O signal	GPIO	1	I/O	21-1/21-2	TSEC2_COL	16-2/16-7
GPIO[1]	General-purpose I/O signal	GPIO	1	I/O	21-1/21-2	TSEC2_TX_ER	16-2/16-7
GPIO[10]	General-purpose I/O signal	GPIO	1	I/O	21-1/21-2	TSEC2_TX_CLK/ TSEC2_GTX_CLK125	16-2/16-7
GPIO[11:14]	General-purpose I/O signal	GPIO	4	I/O	21-1/21-2	TSEC2_TXD[3:0]	16-2/16-7
GPIO[15]	General-purpose I/O signal	GPIO	1	I/O	21-1/21-2	TSEC2_TX_EN	16-2/16-7
GPIO[2]	General-purpose I/O signal	GPIO	1	I/O	21-1/21-2	TSEC2_GTX_CLK	16-2/16-7
GPIO[3]	General-purpose I/O signal	GPIO	1	I/O	21-1/21-2	TSEC2_RX_CLK	16-2/16-7
GPIO[4]	General-purpose I/O signal	GPIO	1	I/O	21-1/21-2	TSEC2_RX_DV	16-2/16-7
GPIO[5:7]	General-purpose I/O signal	GPIO	3	I/O	21-1/21-2	—	—
GPIO[8]	General-purpose I/O signal	GPIO	1	I/O	21-1/21-2	TSEC2_RXD[0]	16-2/16-7
GPIO[9]	General-purpose I/O signal	GPIO	1	I/O	21-1/21-2	TSEC2_RX_ER	16-2/16-7
$\overline{\text{GTM1\_TGATE1}}$	Timer gate 1	Global Timers	1	I/O	5-53/5-55	SD_WP/GPIO[19]	11-1/11-4/ 21-1/21-2
$\overline{\text{GTM1\_TGATE2}}$	Timer gate 2	Global Timers	1	I/O	5-53/5-55	SD_DAT[3]/GPIO[23]	11-1/11-4/ 21-1/21-2
$\overline{\text{GTM1\_TGATE3}}$	Timer gate 3	Global Timers	1	I	5-53/5-55	—	—

Table 2-2. MPC8308 Signal Reference by Alphabetical Order

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
$\overline{\text{GTM1\_TGATE4}}$	Timer gate 4	Global Timers	1	I/O	5-53/5-55	GPIO[15]	21-1/21-2
GTM1_TIN1	Timer in 1	Global Timers	1	I/O	5-53/5-55	$\overline{\text{SD\_CD}}$ /GPIO[18]	11-1/11-4/ 21-1/21-2
GTM1_TIN2	Timer in 2	Global Timers	1	I/O	5-53/5-55	SD_DAT[2]/GPIO[22]	11-1/11-4/ 21-1/21-2
GTM1_TIN3	Timer in 3	Global Timers	1	I	5-53/5-55	—	—
GTM1_TIN4	Timer in 4	Global Timers	1	I	5-53/5-55	—	—
$\overline{\text{GTM1\_TOUT}}[3:4]$	Timer out 3, 4	Global Timers	2	I/O	5-53/5-55	GPIO[9:10]	21-1/21-2
$\overline{\text{GTM1\_TOUT1}}$	Timer out 1	Global Timers	1	I/O	5-53/5-55	SD_DAT[0]/GPIO[20]	11-1/11-4/ 21-1/21-2
$\overline{\text{GTM1\_TOUT2}}$	Timer out 2	Global Timers	1	I/O	5-53/5-55	SD_DAT[1]/GPIO[21]	11-1/11-4/ 21-1/21-2
$\overline{\text{HRESET}}$	Hard reset	System control	1	I/O	4-1/4-1	—	—
IIC_SCL1	I <sup>2</sup> C serial clock 1	I <sup>2</sup> C	1	I/O	17-1/17-3	—	—
IIC_SCL2	I <sup>2</sup> C serial clock 2	I <sup>2</sup> C	1	I/O	17-1/17-3	$\overline{\text{CKSTOP\_IN}}$	—
IIC_SDA1	I <sup>2</sup> C serial data 1	I <sup>2</sup> C	1	I/O	17-1/17-3	—	—
IIC_SDA2	I <sup>2</sup> C serial data 2	I <sup>2</sup> C	1	I/O	17-1/17-3	$\overline{\text{CKSTOP\_OUT}}$	—
$\overline{\text{IRQ}}[0]/\overline{\text{MCP\_IN}}$	External interrupt 0	IPIC	1	I	8-1/8-5	—	—
$\overline{\text{IRQ}}[1]$	External interrupt 1	IPIC	1	I/O	8-1/8-5	$\overline{\text{MCP\_OUT}}$	8-1/8-5
$\overline{\text{IRQ}}[2]$	External interrupt 2	IPIC	1	I/O	8-1/8-5	$\overline{\text{CKSTOP\_OUT}}$	—
$\overline{\text{IRQ}}[3]$	External interrupt 3	IPIC	1	I	8-1/8-5	$\overline{\text{CKSTOP\_IN}}$	—
LA[0:25]	LBC port address	eLBC	26	O	10-2/10-5	—	—
LB_POR_CFG_BOOT_ECC	Enable/Disable ECC for Flash during RCW load	eLBC	1	I	—	TSEC1_TX_ER	16-2/16-7
LBC_PM_REF_10	Status of ECC error during boot loading from flash	eLBC	1	O	—	TSEC1_TX_EN	16-2/16-7
LBCTL	LBC data buffer control	eLBC	1	O	10-2/10-5	—	—
LCLK0	LBC clock	eLBC	1	O	10-2/10-5	—	—
$\overline{\text{LCS}}[0:3]$	LBC chip select 0–3	eLBC	4	O	10-2/10-5	—	—
LD[0:15]	LBC data	eLBC	16	I/O	10-2/10-5	—	—

Table 2-2. MPC8308 Signal Reference by Alphabetical Order

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
LGPL0, LFCLE	LBC UPM general purpose line 0, Flash command latch enable	eLBC	1	O	10-2/10-5	—	—
LGPL1, LFALE	LBC GP line 1, Flash address latch enable	eLBC	1	O	10-2/10-5	—	—
LGPL2, $\overline{\text{LOE}}$ , $\overline{\text{LFRE}}$	LBC GP line 2, LBC output enable, Flash read enable	eLBC	1	O	10-2/10-5	—	—
LGPL3, $\overline{\text{LFWP}}$	LBC GP line 3, Flash write protect	eLBC	1	O	10-2/10-5	—	—
LGPL4, $\overline{\text{LGTA}}$ , LUPWAIT, $\overline{\text{LFRB}}$	LBC GP line 4, Transaction termination, External device wait, Flash ready/busy	eLBC	1	I/O	10-2/10-5	—	—
LGPL5	LBC GP line 5	eLBC	1	O	10-2/10-5	—	—
$\overline{\text{LWE0}}$ , $\overline{\text{LFWEO}}$ , $\overline{\text{LBS0}}$	LBC write enable, Byte (lane) select	eLBC	1	O	10-2/10-5	—	—
$\overline{\text{LWE1}}$ , $\overline{\text{LBS1}}$	LBC write enable, Byte (lane) select	eLBC	1	O	10-2/10-5	—	—
MA[13:0]	DDR address	DDR2	14	O	9-1/9-4	—	—
MBA[2:0]	DDR bank select	DDR2	3	O	9-1/9-4	—	—
$\overline{\text{MCAS}}$	DDR column address strobe	DDR2	1	O	9-1/9-4	—	—
MCK[0:2]	DDR differential clocks	DDR2	3	O	9-1/9-4	—	—
$\overline{\text{MCK}}$ [0:2]	DDR differential clocks	DDR2	3	O	9-1/9-4	—	—
MCKE	DDR clock enable	DDR2	1	O	9-1/9-4	—	—
$\overline{\text{MCP\_OUT}}$	Machine check interrupt output	IPIC	1	O	8-1/8-5	$\overline{\text{IRQ}}[1]$	8-1/8-5
$\overline{\text{MCS}}$ [0:1]	DDR chip select (2/DIMM)	DDR2	2	O	9-1/9-4	—	—
MDM[0:3]	DDR data mask	DDR2	4	O	9-1/9-4	—	—
MDM[8]	DDR data mask	DDR2	1	O	9-1/9-4	—	—
MDQ[0:31]	DDR data	DDR2	32	I/O	9-1/9-4	—	—



Table 2-2. MPC8308 Signal Reference by Alphabetical Order

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
MDQS[0:3]	DDR data strobe	DDR2	4	I/O	9-1/9-4	—	—
MDQS[8]	DDR data strobe	DDR2	1	I/O	9-1/9-4	—	—
MDVAL/LDVAL	Memory debug data valid	Debug	1	I/O	—, 10-2/10-5	SPIMISO	19-1/19-6
MECC[0:7]	DDR ECC data	DDR2	8	I/O	9-1/9-4	—	—
MODT[0:1]	DRAM on-die termination	DDR2	2	O	9-1/9-4	—	—
$\overline{\text{MRAS}}$	DDR row address strobe	DDR2	1	O	9-1/9-4	—	—
MSRCID0/LSRCID0	Memory debug source ID	Debug	1	O	—, 10-2/10-5	UART_SOUT[1]	18-1/18-3
MSRCID1/LSRCID1	Memory debug source ID	Debug	1	I/O	—, 10-2/10-5	UART_SIN[1]	18-1/18-3
MSRCID2/LSRCID2	Memory debug source ID	Debug	1	I/O	—, 10-2/10-5	UART_SOUT[2]	18-1/18-3
MSRCID3/LSRCID3	Memory debug source ID	Debug	1	I/O	—, 10-2/10-5	UART_SIN[2]	18-1/18-3
MSRCID4/LSRCID4	Memory debug source ID	Debug	1	O	—, 10-2/10-5	SPIMOSI	19-1/19-6
MVREF	DDR2 DRAM reference	DDR2	1	PWR	9-1/9-4	—	—
$\overline{\text{MWE}}$	DDR write enable	DDR2	1	O	9-1/9-4	—	—
$\overline{\text{PORESET}}$	Power on reset	System control	1	I	4-1/4-1	—	—
$\overline{\text{QUIESCE}}$	Quiescent state	Misc	1	O	5-64/5-69	—	—
RTC_PIT_CLOCK	32.768 KHz clock input to RTC	Clocks	1	I	15-1/15-4	—	—
RXA	Serial receiver, lane A, positive data	PCI Express PHY	1	I	15-1/15-2	—	—
$\overline{\text{RXA}}$	Serial receiver, lane A, negative data (complement)	PCI Express PHY	1	I	15-1/15-2	—	—
$\overline{\text{SD\_CD}}$	Card detection signal	eSDHC	1	I/O	11-1/11-4	GTM1_TIN1/ GPIO[18]	5-53/5-55/ 21-1/21-2
SD_CLK	eSDHC clock out	eSDHC	1	I/O	11-1/11-4	GPIO[16]	21-1/21-2
SD_CMD	eSDHC command/response signals	eSDHC	1	I/O	11-1/11-4	GPIO[17]	21-1/21-2
SD_DAT[0]	Data signal 0	eSDHC	1	I/O	11-1/11-4	$\overline{\text{GTM1\_TOUT1}}$ / GPIO[20]	5-53/5-55/ 21-1/21-2

Table 2-2. MPC8308 Signal Reference by Alphabetical Order

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
SD_DAT[1]	Data signal 1	eSDHC	1	I/O	11-1/11-4	$\overline{\text{GTM1\_TOUT2}}$ / GPIO[21]	5-53/5-55/ 21-1/21-2
SD_DAT[2]	Data signal 2	eSDHC	1	I/O	11-1/11-4	GTM1_TIN2/ GPIO[22]	5-53/5-55/ 21-1/21-2
SD_DAT[3]	Data signal 3	eSDHC	1	I/O	11-1/11-4	$\overline{\text{GTM1\_TGATE2}}$ / GPIO[23]	5-53/5-55/ 21-1/21-2
SD_IMP_CAL_RX	Receiver impedance control signal	PCI Express PHY	1	I	15-1/15-2	—	—
SD_IMP_CAL_TX	Transmitter impedance control signal	PCI Express PHY	1	I	15-1/15-2	—	—
SD_PLL_TPA_ANA	Analog test point for SerDes PLL testing	PCI Express PHY	1	O	—	—	—
SD_PLL_TPD	Digital test point for SerDes PLL testing	PCI Express PHY	1	O	—	—	—
SD_REF_CLK	SerDes PLL reference clock	PCI Express PHY	1	I	15-1/15-2	—	—
$\overline{\text{SD\_REF\_CLK}}$	SerDes PLL reference clock (complement)	PCI Express PHY	1	I	15-1/15-2	—	—
SD_WP	Write protection signal	eSDHC	1	I/O	11-1/11-4	$\overline{\text{GTM1\_TGATE1}}$ / GPIO[19]	5-53/5-55/ 21-1/21-2
SDAVDD	Analog supply for SerDes PLL	PCI Express PHY	1	PWR	—	—	—
SDAVSS	Analog ground for SerDes PLL	PCI Express PHY	1	GND	—	—	—
SPICLK	SPI clock	SPI	1	I/O	19-1/19-6	—	—
SPIMISO	SPI master-in slave-out	SPI	1	I/O	19-1/19-6	MDVAL/LDVAL	—, 10-2/10-5
SPIMOSI	SPI master-out slave-in	SPI	1	I/O	19-1/19-6	MSRCID4/LSRCID4	—, 10-2/10-5
$\overline{\text{SPISEL}}$	SPI slave select	SPI	1	I	19-1/19-6	—	—
$\overline{\text{SRESET}}$	Soft reset	System control	1	I/O	4-1/4-1	—	—
SYS_CLK_IN	Clock input	Clocks	1	I	4-2/4-2	—	—
TCK	Test clock	JTAG	1	I	20-1/20-2	—	—
TDI	Test data in	JTAG	1	I	20-1/20-2	—	—

Table 2-2. MPC8308 Signal Reference by Alphabetical Order

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
TDO	Test data out	JTAG	1	O	<a href="#">20-1/20-2</a>	—	—
TEST_MODE	Internal test signal <b>Note:</b> "This pin must always be tied to VSS"	Test	1	I	—	—	—
THERM0	Thermal resistor access 0	Misc	1	I	—	—	—
TMS	Test mode select	JTAG	1	I	<a href="#">20-1/20-2</a>	—	—
$\overline{\text{TRST}}$	Test reset	JTAG	1	I	<a href="#">20-1/20-2</a>	—	—
TSEC_TMR_ALARM[1]	1588 timer alarm-out 1	eTSEC	1	O	<a href="#">16-2/16-7</a>	—	—
TSEC_TMR_ALARM[2]	1588 timer alarm-out 2	eTSEC	1	I/O	<a href="#">16-2/16-7</a>	GPIO[14]	<a href="#">21-1/21-2</a>
TSEC_TMR_CLK	1588 clock-in	eTSEC	1	I/O	<a href="#">16-2/16-7</a>	GPIO[8]	<a href="#">21-1/21-2</a>
TSEC_TMR_GCLK	1588 clock-out	eTSEC	1	O	<a href="#">16-2/16-7</a>	—	—
TSEC_TMR_PP[1:2]	1588 timer pulse-out 1, 2	eTSEC	2	O	<a href="#">16-2/16-7</a>	—	—
TSEC_TMR_PP[3]	1588 timer pulse-out 3	eTSEC	1	I/O	<a href="#">16-2/16-7</a>	GPIO[13]	<a href="#">21-1/21-2</a>
TSEC_TMR_TRIG[1:2]	1588 trigger-in 1, 2	eTSEC	2	I/O	<a href="#">16-2/16-7</a>	GPIO[11:12]	<a href="#">21-1/21-2</a>
TSEC1_COL	eTSEC1 collision detect	eTSEC1	1	I	<a href="#">16-2/16-7</a>	—	—
TSEC1_CRS	eTSEC1 carrier sense	eTSEC1	1	I	<a href="#">16-2/16-7</a>	—	—
TSEC1_GTX_CLK	eTSEC1 transmit clock out	eTSEC1	1	O	<a href="#">16-2/16-7</a>	—	—
TSEC1_GTX_CLK125	Gigabit reference clock	eTSEC1	1	I	<a href="#">16-2/16-7</a>	TSEC1_TX_CLK	<a href="#">16-2/16-7</a>
TSEC1_MDC	Ethernet management data clock	Ethernet Management	1	O	<a href="#">16-2/16-7</a>	—	—
TSEC1_MDIO	Ethernet management data in/out	Ethernet Management	1	I/O	<a href="#">16-2/16-7</a>	—	—
TSEC1_RX_CLK	eTSEC1 receive clock	eTSEC1	1	I	<a href="#">16-2/16-7</a>	—	—
TSEC1_RX_DV	eTSEC1 receive data valid	eTSEC1	1	I	<a href="#">16-2/16-7</a>	—	—

Table 2-2. MPC8308 Signal Reference by Alphabetical Order

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
TSEC1_RX_ER	eTSEC1 receiver error	eTSEC1	1	I	16-2/16-7	—	—
TSEC1_RXD[3:0]	eTSEC1 receive data 3–0	eTSEC1	4	I	16-2/16-7	—	—
TSEC1_TMR_RX_ESFD	Transmit external start of frame delimiter (for eTSEC1)	eTSEC	1	I/O	16-2/16-7	GPIO[3]	21-1/21-2
TSEC1_TMR_TX_ESFD	Receive external start of frame delimiter (for eTSEC1)	eTSEC	1	I/O	16-2/16-7	GPIO[4]	21-1/21-2
TSEC1_TX_CLK	eTSEC1 transmit clock in	eTSEC1	1	I	16-2/16-7	TSEC1_GTX_CLK125	16-2/16-7
TSEC1_TX_EN	eTSEC1 transmit enable	eTSEC1	1	O	16-2/16-7	LBC_PM_REF_10	—
TSEC1_TX_ER	eTSEC1 transmit error	eTSEC1	1	I/O	16-2/16-7	LB_POR_CFG_BOOT_ECC	—
TSEC1_TXD[3:0]	eTSEC1 transmit data 3–0	eTSEC1	4	I/O	16-2/16-7	CFG_RESET_SOURCE[0:3]	4-1/4-1
TSEC2_COL	eTSEC2 collision detect	eTSEC2	1	I/O	16-2/16-7	GPIO[0]	21-1/21-2
TSEC2_CRS	eTSEC2 carrier sense	eTSEC2	1	I/O	16-2/16-7	GPIO[0]	21-1/21-2
TSEC2_GTX_CLK	eTSEC2 transmit clock out	eTSEC2	1	I/O	16-2/16-7	GPIO[2]	21-1/21-2
TSEC2_GTX_CLK125	Gigabit reference clock	eTSEC2	1	I/O	16-2/16-7	GPIO[10]/TSEC2_TX_CLK	21-1/21-2/
TSEC2_RX_CLK	eTSEC2 receive clock	eTSEC2	1	I/O	16-2/16-7	GPIO[3]	21-1/21-2
TSEC2_RX_DV	eTSEC2 receive data valid	eTSEC2	1	I/O	16-2/16-7	GPIO[4]	21-1/21-2
TSEC2_RX_ER	eTSEC2 receiver error	eTSEC2	1	I/O	16-2/16-7	GPIO[9]	21-1/21-2
TSEC2_RXD[0]	eTSEC2 receive data 0	eTSEC2	1	I/O	16-2/16-7	GPIO[8]	21-1/21-2
TSEC2_RXD[3:1]	eTSEC2 receive data 3–1	eTSEC2	3	I/O	16-2/16-7	GPIO[5:7]	21-1/21-2

Table 2-2. MPC8308 Signal Reference by Alphabetical Order

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
TSEC2_TMR_RX_ESFD	Receive external start of frame delimiter (for eTSEC2)	eTSEC	1	I/O	16-2/16-7	GPIO[1]	21-1/21-2
TSEC2_TMR_TX_ESFD	Transmit external start of frame delimiter (for eTSEC2)	eTSEC	1	I/O	16-2/16-7	GPIO[2]	21-1/21-2
TSEC2_TX_CLK	eTSEC2 transmit clock in	eTSEC2	1	I/O	16-2/16-7	GPIO[10]/ TSEC2_GTX_CLK125	21-1/21-2/ —
TSEC2_TX_EN	eTSEC2 transmit enable	eTSEC2	1	I/O	16-2/16-7	GPIO[15]	21-1/21-2
TSEC2_TX_ER	eTSEC2 transmit error	eTSEC2	1	I/O	16-2/16-7	GPIO[1]	21-1/21-2
TSEC2_TXD[3:0]	eTSEC2 transmit data 3–0	eTSEC2	4	I/O	16-2/16-7	GPIO[11:14]	21-1/21-2
TXA	Serial transmitter, lane A, positive data	PCI Express PHY	1	O	15-1/15-2	—	—
$\overline{\text{TXA}}$	Serial transmitter, lane A, negative data (complement)	PCI Express PHY	1	O	15-1/15-2	—	—
UART_SIN[1]	DUART serial data in	DUART	1	I/O	18-1/18-3	MSRCID1/LSRCID1	—, 10-2/10-5
UART_SIN[2]	DUART serial data in	DUART	1	I/O	18-1/18-3	MSRCID3/LSRCID3	—, 10-2/10-5
UART_SOUT[1]	DUART serial data out	DUART	1	O	18-1/18-3	MSRCID0/LSRCID0	—, 10-2/10-5
UART_SOUT[2]	DUART serial data out	DUART	1	O	18-1/18-3	MSRCID2/LSRCID2	—, 10-2/10-5
USBDR_CLK	Clocking signal for ULPI PHY interface	USB	1	I	13-2/13-3	—	—
USBDR_DIR	Direction of data bus	USB	1	I	13-2/13-3	—	—
USBDR_NXT	Nest data	USB	1	I	13-2/13-3	—	—
USBDR_PCTL[0:1]	Port control 0–1	USB	2	O	13-2/13-3	—	—
USBDR_PWR_FAULT	USB VBus power fault	USB	1	I	13-2/13-3	—	—
USBDR_STP	End of a transfer on the bus	USB	1	O	13-2/13-3	—	—

Table 2-2. MPC8308 Signal Reference by Alphabetical Order

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
USBD <sub>R</sub> _TXDRXD[0:7]	Data bit 0–7	USB	8	I/O	13-2/13-3	—	—
XCOREVDD[0:2]	SerDes transceiver core supply	PCI Express PHY	3	PWR	—	—	—
XCOREVSS[0:2]	SerDes transceiver core ground	PCI Express PHY	3	GND	—	—	—
XPADVDD[0:1]	SerDes transceiver pad supply	PCI Express PHY	2	PWR	—	—	—
XPADVSS[0:1]	SerDes transceiver pad ground	PCI Express PHY	2	GND	—	—	—

## 2.2 Output Signal States During Reset

When a system reset is recognized ( $\overline{\text{PORESET}}$  or  $\overline{\text{HRESET}}$  are asserted), the device aborts all current internal and external transactions (with the exception of RTC) 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 device ignores most input signals (except for the reset configuration signals) and drives most of the output-only signals to an inactive state. Table 2-3 shows the states of the output-only signals.

Table 2-3. Output Signal States During System Reset

Interface	Signal	State During Reset
MDM[0:3]	DDR data mask	High-Z
MDM[8]	DDR data mask	High-Z
MBA[2:0]	DDR bank select	High-Z
MA[13:0]	DDR address	High-Z
$\overline{\text{MWE}}$	DDR write enable	High-Z
$\overline{\text{MRAS}}$	DDR row address strobe	High-Z
$\overline{\text{MCAS}}$	DDR column address strobe	High-Z
$\overline{\text{MCS}}$ [0:1]	DDR chip select (2/DIMM)	High-Z
MCKE	DDR clock enable	Driven Low
MCK[0:2]	DDR differential clocks	Low
$\overline{\text{MCK}}$ [0:2]	DDR differential clocks	Low
MODT[0:1]	DRAM on-die termination	Driven Low
UART_SOUT[1:2]	DUART serial data out	High-Z
LA[0:25]	LBC port address	Active—used to load reset configuration word

Table 2-3. Output Signal States During System Reset (continued)

Interface	Signal	State During Reset
$\overline{\text{LCS}}[0]$	LBC chip select 0	Active—used to load reset configuration word
$\overline{\text{LCS}}[1:3]$	LBC chip select	High
$\overline{\text{LWE}}[0:1]$	LBC write enable	High
LBCTL	LBC data buffer control	Active—used to load reset configuration word
$\overline{\text{LOE}}/\text{LGPL}2$	LBC output enable/GP line 2	Active—used to load reset configuration word
LCLK0	LBC clock 0	High-Z
LGPL[0:1], LGPL[3:5]	LBC UPM General purpose line	High
TSEC1_MDC	Ethernet management data clock	Low
TSEC1_GTX_CLK	eTSEC1 transmit clock out	Low
TSEC1_TXD[3:0]	eTSEC1 transmit data 3–0	Low
TSEC1_TX_EN	eTSEC1 transmit enable	Low
TSEC1_TX_ER	eTSEC1 transmit error	Low
TSEC2_TXD[3:0]	eTSEC2 transmit data 3–0	Low
TSEC2_TX_EN	eTSEC2 transmit enable	Low
TSEC2_TX_ER	eTSEC2 transmit error	Low
TSEC_TMR_GCLK	1588 clock-out	Low
TSEC_TMR_PP[1:3]	1588 timer pulse-out 1, 2, 3	Low
TSEC_TMR_ALARM[1:2]	1588 timer alarm-out 1, 2	Low
SD_CLK	eSDHC clock out	Low
SD_PLL_TPD	Digital test point for SerDes PLL testing	High
TDO	Test data out	High-Z
$\overline{\text{QUIESCE}}$	Quiescent state	High
$\overline{\text{GTM1\_TOUT}}[3:4]$	Timer out 3, 4	High
USBDR_PCTL[0:1]	Port control 0–1	Low
USBDR_STP	End of a transfer on the bus	High





## Chapter 3

# Memory Map

This chapter describes the MPC8308 memory map. The internal memory-mapped registers are described, including a complete listing of all memory-mapped registers with cross references to the sections detailing descriptions of each.

### 3.1 Internal Memory-Mapped Registers

All of the memory-mapped registers in the device are contained within a 1-Mbyte address region. To allow for flexibility, the base address of the memory-mapped registers is re-locatable in the local address space. The local address map location of this register block is controlled by the internal memory-mapped registers base address register (IMMRBAR). See [Section 5.1.4.1, “Internal Memory Map Registers Base Address Register \(IMMRBAR\),”](#) for more information. The default value for IMMRBAR is 0xFF40\_0000.

### 3.2 Accessing IMMR Memory from the Local Processor

When the local e300 processor is used to configure IMMR space, the IMMR 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 the 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 followed immediately 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.

### 3.3 IMMR Address Map

[Table 3-1](#) lists the location of the functional block base addresses for the entire IMMRBAR space. Unless stated otherwise in a particular block, all accesses to and from the memory-mapped registers must be made with 32-bit accesses. There is no support for accesses of sizes other than 32 bits.

Reading from address locations that appear as reserved in the memory map table is not guaranteed to return predictable data. Writing to address locations that appear as reserved in the memory map table is not allowed and could lead to unpredictable behavior of the device. Reserved bits in non-reserved registers are read as zero unless the reset value of those bits is different due to internal logic considerations.

When writing to registers with reserved bits, those reserved bits should be cleared. By doing so, existing software would be able to run on a future modified device in which some reserved bits were allocated for enhanced modes. This would allow for maintaining the legacy functionality when set to zero.

## Memory Map

In certain specific cases, reserved bits should not be cleared but should keep their reset value. Thus, the software should perform a ‘read-modify-write’ and make sure that it does not change the reset value of those bits. The description of the specific bits indicate when this is needed.

Cross-references are provided to the IMMRBAR maps for each individual block. A complete listing of all registers is provided in [Appendix A, “Complete List of Configuration, Control, and Status Registers.”](#)

**Table 3-1. IMMR Memory Map**

Block Base Address	Block	Actual Size	Window	Section/ Page
0x0_0000–0x0_01FF	System configuration	512 bytes	512 bytes	<a href="#">5.2.1/5-15</a>
0x0_0200–0x0_02FF	Watchdog timer	16 bytes	256 bytes	<a href="#">5.3.4/5-33</a>
0x0_0300–0x0_03FF	Real time clock	32 bytes	256 bytes	<a href="#">15.6/15-5</a>
0x0_0400–0x0_04FF	Periodic interval timer	32 bytes	256 bytes	<a href="#">5.5.5/5-48</a>
0x0_0500–0x0_05FF	Global timers module	64 bytes	256 bytes	<a href="#">5.6.5/5-57</a>
0x0_0600–0x0_06FF	Reserved	—	256 bytes	—
0x0_0700–0x0_07FF	Integrated programmable interrupt controller (IPIC)	128 bytes	256 bytes	<a href="#">8.5/8-5</a>
0x0_0800–0x0_08FF	System arbiter	30 bytes	256 bytes	<a href="#">6.2/6-2</a>
0x0_0900–0x0_09FF	Reset module	44 bytes	256 bytes	<a href="#">4.5.1/4-25</a>
0x0_0A00–0x0_0AFF	Clock module	44 bytes	256 bytes	<a href="#">4.5.2/4-29</a>
0x0_0B00–0x0_0BFF	Power management control module	20 bytes	256 bytes	<a href="#">5.7.2/5-69</a>
0x0_0C00–0x0_0CFF	GPIO	24 bytes	256 bytes	<a href="#">21.3/21-2</a>
0x0_0D00–0x0_1FFF	Reserved	—	4.8 Kbytes	—
0x0_2000–0x0_2FFF	DDR2 memory controller	3.8 Kbytes	4 Kbytes	<a href="#">9.4/9-9</a>
0x0_3000–0x0_30FF	I <sup>2</sup> C controller 1	24 bytes	256 bytes	<a href="#">17.3/17-4</a>
0x0_3100–0x0_31FF	I <sup>2</sup> C controller 2	24 bytes	256 bytes	
0x0_3200–0x0_44FF	Reserved	—	4.8 Kbytes	—
0x0_4500–0x0_46FF	DUART (UART1 and UART2)	18 bytes × 2	4 Kbytes	<a href="#">18.3/18-3</a>
0x0_4700–0x0_4FFF	Reserved	—	2.3 Kbytes	—
0x0_5000–0x0_5FFF	eLBC	224 bytes	4 Kbytes	<a href="#">10.3/10-7</a>
0x0_6000–0x0_6FFF	Reserved	—	4 Kbytes	—
0x0_7000–0x0_7FFF	SPI	24 bytes	4 Kbytes	<a href="#">19.3/19-7</a>
0x0_8000–0x0_8FFF	Reserved	—	4 Kbytes	—
0x0_9000–0x0_9FFF	PCI Express	4 Kbytes	4 Kbytes	<a href="#">14.3/14-5</a>
0x0_A000–0x2_2FFF	Reserved	4 Kbytes	102.4 Kbytes	—
0x2_3000–0x2_3FFF	USB dual-role (DR) controller	4 Kbytes	4 Kbytes	<a href="#">13.3/13-4</a>

Table 3-1. IMMR Memory Map (continued)

Block Base Address	Block	Actual Size	Window	Section/ Page
0x2_4000–0x2_4FFF	eTSEC1	4 Kbytes	4 Kbytes	16.5/16-9
0x2_5000–0x2_5FFF	eTSEC2	4 Kbytes	4 Kbytes	
0x2_6000–0x2_BFFF	Reserved	—	28.6 Kbytes	—
0x2_C000–0x2_DFFF	DMAC	8 Kbytes	8 Kbytes	12.2/12-2
0x2_E000–0x2_FFFF	eSDHC	8 Kbytes	8 Kbytes	11.4/11-4
0x2_F000–0xE_2FFF	Reserved	—	737.2 Kbytes	—
0xE_3000–0xE_30FF	SerDes	—	256 bytes	15.3/15-3
0xE_3100–0xF_FFFF	Reserved	—	118.5 Kbytes	—



# Chapter 4

## Reset, Clocking, and Initialization

The reset, clocking, and control signals offer many options for operating the device. Various modes and features can be configured during hard reset or power-on reset. Most configurable features are loaded to the device through a reset configuration word, and a few device signals are used as reset configuration inputs during the reset sequence.

### 4.1 External Signals

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

#### 4.1.1 Reset Signals

Table 4-1 describes the reset signals of the device. Section 4.3.2, “Reset Configuration Words,” describes the signals that also function as reset configuration signals.

**Table 4-1. System Control Signals**

Signal	I/O	Description
$\overline{\text{PORESET}}$	I	Power-on reset. Initiates the power-on reset flow that resets the device and configures various attributes of the device, including its clock modes.
		<b>State Meaning</b> Asserted—An external agent has triggered a power-on reset sequence. Negated—No power-on reset.
		<b>Timing</b> For timing information, see <i>PowerQUICC II Pro MPC8308 Hardware Specification</i> .
		<b>Reset State</b> Always input.
$\overline{\text{HRESET}}$	I/O	Hard reset. Causes the device to abort all current internal and external transactions and set most registers to their default values. $\overline{\text{HRESET}}$ can be asserted completely asynchronously with respect to all other signals. The device can detect an external assertion of $\overline{\text{HRESET}}$ while the device is not asserting hard reset. $\overline{\text{HRESET}}$ is an open-drain signal.
		<b>State Meaning</b> Asserted—An external agent or internal hardware has triggered a hard reset. The internal hardware drives $\overline{\text{HRESET}}$ until the sequence completes. Negated—No hard reset.
		<b>Timing</b> Assertion—Occur at any time, asynchronously to any clock. Negation—Must be asserted for at least 32 SYS_CLK_IN cycles.
		<b>Requirements</b> An open-drain signal. An external pull-up is required.
		<b>Reset State</b> Output, driven low during power-on and hard reset flows. High impedance after reset flow completes.

Table 4-1. System Control Signals (continued)

Signal	I/O	Description	
CFG_RESET_SOURCE[0:3]	I	Reset configuration word source selection. These signals are on device pins that have other functions when the device is not in reset. They are sampled during the assertion of $\overline{\text{PORESET}}$ to determine the interface from which the device loads the reset configuration words.	
		<b>State Meaning</b>	See Section 4.3.1.1, "Reset Configuration Word Source."
		<b>Timing</b>	These signals are sampled during the assertion of $\overline{\text{PORESET}}$ after a stable clock is supplied ( $\overline{\text{PORESET}}$ flow) and must be pulled high or low by external resistors as long as $\overline{\text{HRESET}}$ is asserted.
		<b>Requirements</b>	During $\overline{\text{PORESET}}$ and $\overline{\text{HRESET}}$ flows, all other signal drivers connected to these signals must be in the high-impedance state. For proper resistor values to pull reset configuration signals high or low, see <i>PowerQUICC II Pro MPC8308 Hardware Specification</i> .
		<b>Reset State</b>	Input during power-on and hard reset flows. Functional signal after reset flow completes.
$\overline{\text{SRESET}}$	I	This signal generates a high priority interrupt to the e300 core	
		<b>State Meaning</b>	Asserted - An external agent has triggered a soft reset interrupt. Negated - No soft reset interrupt.
		<b>Timing</b>	Assertion - Occur at any time, asynchronously to any clock Negation - Must be asserted for at least 2 SYS_CLK_IN cycles.
		<b>Reset State</b>	High during power-on and hard reset flows.

## 4.1.2 Clock Signals

In Table 4-2, some clock signals are specific to blocks within the device. Although some of their functionality is described in Section 4.4, "Clocking," they are defined in detail in their respective chapters. See Figure 4-7 for the internal distribution of clocks in the device.

Table 4-2. External Clock Signals

Signal	I/O	Description	
SYS_CLK_IN	I	System clock. SYS_CLK_IN is the primary input clock.	
		<b>Timing</b>	Assertion/Negation—For timing information, see <i>PowerQUICC II Pro MPC8308 Hardware Specification</i> .
		<b>Reset State</b>	Always input.
TSEC <sub>n</sub> _RX_CLK	I	Ethernet Receive clock	
		<b>Timing</b>	Assertion/Negation—For timing information, see <i>PowerQUICC II Pro MPC8308 Hardware Specification</i> .
		<b>Reset State</b>	Always input.

Table 4-2. External Clock Signals (continued)

Signal	I/O	Description	
TSEC <sub>n</sub> _TX_CLK/ TSEC <sub>n</sub> _GTX_CLK125	I	Ethernet Transmit Clock. TSEC <sub>n</sub> _TX_CLK is used in the MII mode. TSEC <sub>n</sub> _GTX_CLK125 is the 125-MHz clock used in the RGMII mode.	
		<b>Timing</b>	Assertion/Negation—For timing information, see <i>PowerQUICC II Pro MPC8308 Hardware Specification</i> .
		<b>Reset State</b>	Always input.

## 4.2 Functional Description

This section describes the various ways to reset the device, the power-on reset configurations, and clocking.

### 4.2.1 Reset Operations

The device has the following inputs to the reset logic:

- Power-on reset ( $\overline{\text{PORESET}}$ )
- External hard reset ( $\overline{\text{HRESET}}$ )
- Software watchdog reset
- System bus monitor reset
- Checkstop reset
- Software hard reset

All of these reset sources are fed into the reset controller and, depending on the source of the reset, different actions are taken. The reset status register, described in [Section 4.5.1.3, “Reset Status Register \(RSR\),”](#) indicates the last source to cause a reset.

#### 4.2.1.1 Reset Causes

[Table 4-3](#) describes reset causes.

Table 4-3. Reset Causes

Name	Description
Power-on reset ( $\overline{\text{PORESET}}$ )	Input signal. Asserting this signal initiates the power-on reset flow that resets the entire device except RTC and configures various attributes of the device including its clock modes.
Hard reset ( $\overline{\text{HRESET}}$ )	A bidirectional I/O signal. The device can detect an external assertion of $\overline{\text{HRESET}}$ only while it is not asserting hard reset. $\overline{\text{HRESET}}$ is an open-drain signal.
Soft reset ( $\overline{\text{SRESET}}$ )	This is a high-priority interrupt to the e300 core that is generated by an external signal.
Software watchdog reset	After the device watchdog counts to zero, a software watchdog reset is signaled. The enabled software watchdog event then generates an internal hard reset sequence.
System bus monitor reset	After the device CSB bus monitor reaches a timeout condition, a bus monitor reset is asserted. The enabled bus monitor event then generates an internal hard reset sequence.

Table 4-3. Reset Causes (continued)

Name	Description
Checkstop reset	If the core enters checkstop state and the checkstop reset is enabled (RMR[CSRE] = 1), the checkstop reset is asserted. The enabled checkstop event then generates an internal hard reset sequence.
Software hard reset	A hard reset sequence can be initialized by writing to a memory-mapped register (RCR).

#### 4.2.1.2 Reset Actions

The reset control logic determines the cause of reset, synchronizes it if necessary, and resets the appropriate internal hardware. Each reset flow has a different impact on the device logic:

- Power-on reset has the greatest impact, resetting the entire device, including clock logic and error capture registers. This does not reset the RTC module. For more information on RTC reset sequence, see [Chapter 15, “Real Time Clock \(RTC\) Module.”](#)
- Hard reset resets the entire device, excluding RTC module, clock logic, and error capture registers.

The memory controller, system protection logic, interrupt controller, and I/O signals are initialized only on hard reset. A SRESET causes a high-priority interrupt to the e300 core.

Table 4-4 identifies the reset actions for each reset source.

Table 4-4. Reset Actions

Action	Reset Source		
	Power-On Reset	External Hard Reset Software Watchdog Bus Monitor Checkstop Software Hard Reset	$\overline{\text{SRESET}}$
Resets: PLLs, clocks, and error capture registers	Yes	No	No
Resets: DDR controller, LBC, I/O multiplexors, GTM, PIT, GPIO, system configuration, and local access windows	Yes	Yes	No
Resets other internal logic	Yes	Yes	No
Reset configuration words loaded	Yes	Yes	No
$\overline{\text{HRESET}}$ driven by SoC	Yes	Yes	No
Hard reset to e300 core	Yes	Yes	No
High priority interrupt to the e300 core	No	No	Yes
RTC	No	No	No



## 4.2.2 Power-On Reset Flow

Assertion of the  $\overline{\text{PORESET}}$  external signal initiates the power-on reset flow.  $\overline{\text{PORESET}}$  should be asserted externally for at least 32 input clock cycles after stable external power to the device is applied.

Directly after the negation of  $\overline{\text{PORESET}}$ , the device starts the configuration process. The device asserts  $\overline{\text{HRESET}}$  throughout the power-on reset process, including configuration. Configuration time varies according to the configuration source and  $\text{SYS\_CLK\_IN}$  frequency. Initially, the reset configuration inputs are sampled to determine the configuration source. Next, the device starts loading the reset configuration words. The system PLL begins to lock according to the clock mode values in the reset configuration word low. When the system PLL is locked, the clock unit starts distributing clock signals in the device. At this stage, the core PLL begins to lock. When it is locked and the reset configuration words are loaded,  $\overline{\text{HRESET}}$  is released.

The detailed power-on reset (POR) flow for the device is as follows:

1. Power is applied to meet the specifications in the device data sheet.
2. The system asserts  $\overline{\text{PORESET}}$  and  $\overline{\text{TRST}}$ , causing all registers to be initialized to their default states and most I/O drivers to be released to high-impedance.
3. The system applies a stable  $\text{SYS\_CLK\_IN}$  signal and stable reset configuration inputs ( $\text{CFG\_RESET\_SOURCE}$ ).
4. The system negates  $\overline{\text{PORESET}}$  after at least 32 stable  $\text{SYS\_CLK\_IN}$  clock cycles.
5. The device samples the reset configuration input signals to determine the reset configuration words source.
6. The device starts loading the reset configuration words.  
Loading time depends on the reset configuration word source.
7. When the reset configuration word low is loaded, the system PLL begins to lock.  
When the system PLL is locked, *csb\_clk* is supplied to the core PLL.
8. The core PLL begins to lock.
9. The device drives  $\overline{\text{HRESET}}$  asserted until the e300 PLL is locked and the reset configuration words are loaded.
10. The user optionally negates  $\overline{\text{HRESET}}$  if it was not negated earlier.  
JTAG logic must always be initialized by asserting  $\overline{\text{TRST}}$ . If the JTAG signals are not used,  $\overline{\text{TRST}}$  should be connected directly to  $\overline{\text{PORESET}}$ .  $\overline{\text{TRST}}$  must not remain asserted after the negation of  $\overline{\text{PORESET}}$ .
11. The internal reset to the core and the rest of the logic is negated. I/O drivers are enabled.
12. The device stops driving  $\overline{\text{HRESET}}$ . The reset to the e300 core is negated and the core is enabled.  
The boot sequencer, if enabled, is released, causing it to load configuration data from serial ROMs, as described in [Section 17.4.5, “Boot Sequencer Mode.”](#)
13. If the e300 core is required to proceed, the boot sequencer should enable boot vector fetch by clearing  $\text{ACR}[\text{COREDIS}]$  as described in [Section 6.2.1, “Arbiter Configuration Register \(ACR\).”](#)
14. The boot vector fetch by the core can proceed, if enabled.

The device is now in its ready state.

Figure 4-1 shows a timing diagram of the power-on reset flow.

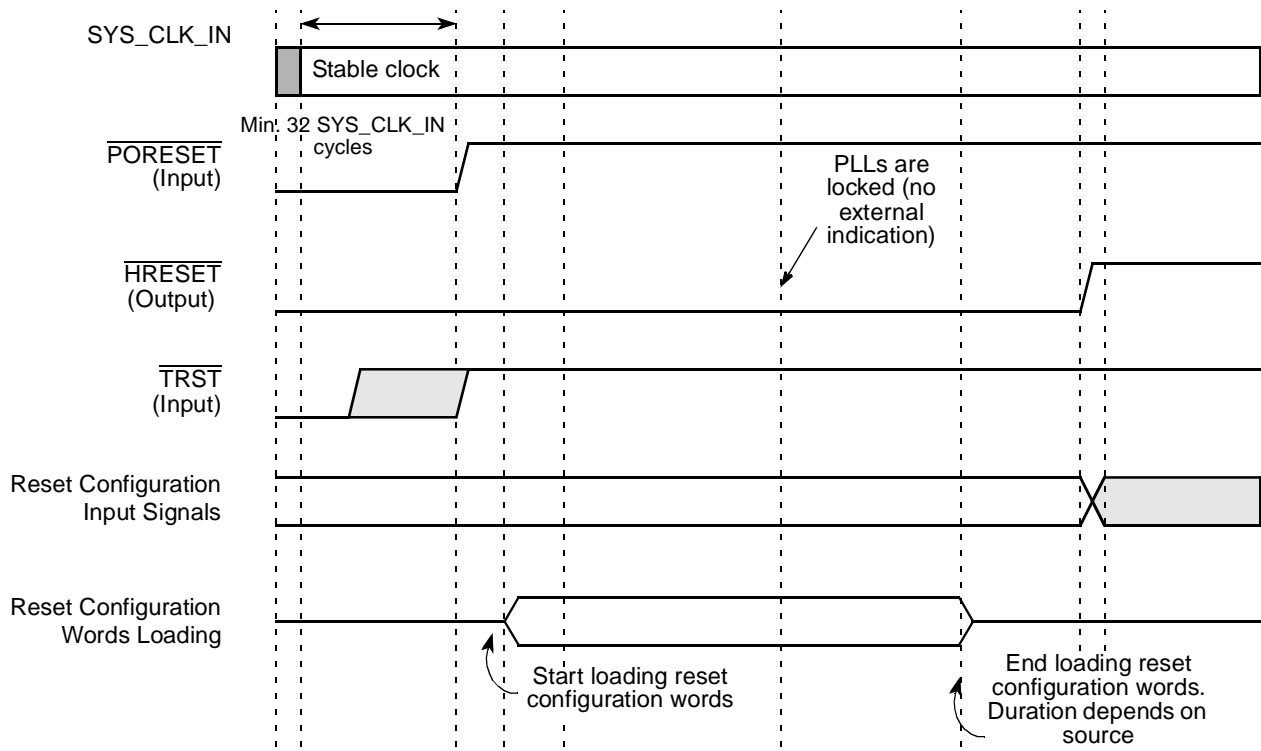


Figure 4-1. Power-On Reset Flow

#### NOTE

It takes about 150 SYS\_CLK\_IN cycles between the negation of PORESET and access to reset configuration word.

### 4.2.3 Hard Reset Flow

The  $\overline{\text{HRESET}}$  signal is initiated externally by asserting  $\overline{\text{HRESET}}$  or internally when the device detects a reason to generate an internal hard reset sequence. In both cases, the device continues asserting  $\overline{\text{HRESET}}$  throughout the  $\overline{\text{HRESET}}$  state. The hard reset sequence time varies according to the configuration source and SYS\_CLK\_IN frequency. The reset configuration input signal (CFG\_RESET\_SOURCE) is not sampled by hard reset (only by power-on reset), so the device immediately starts loading the reset configuration words and configures the device as explained in Section 4.3.3, "Loading the Reset Configuration Words." After the configuration sequence completes, the device releases the  $\overline{\text{HRESET}}$  signal and exits the  $\overline{\text{HRESET}}$  state. An external pull-up resistor should negate the signals. After negation is detected, a 16-cycle period is taken before testing for the presence of an external (hard) reset.

#### NOTE

Because the device does not sample the reset configuration input signals (CFG\_RESET\_SOURCE) during a hard reset flow, setting a new value on those signals (other than that set during power-on reset) has no effect.

Figure 4-2 shows a timing diagram of the hard reset flow.

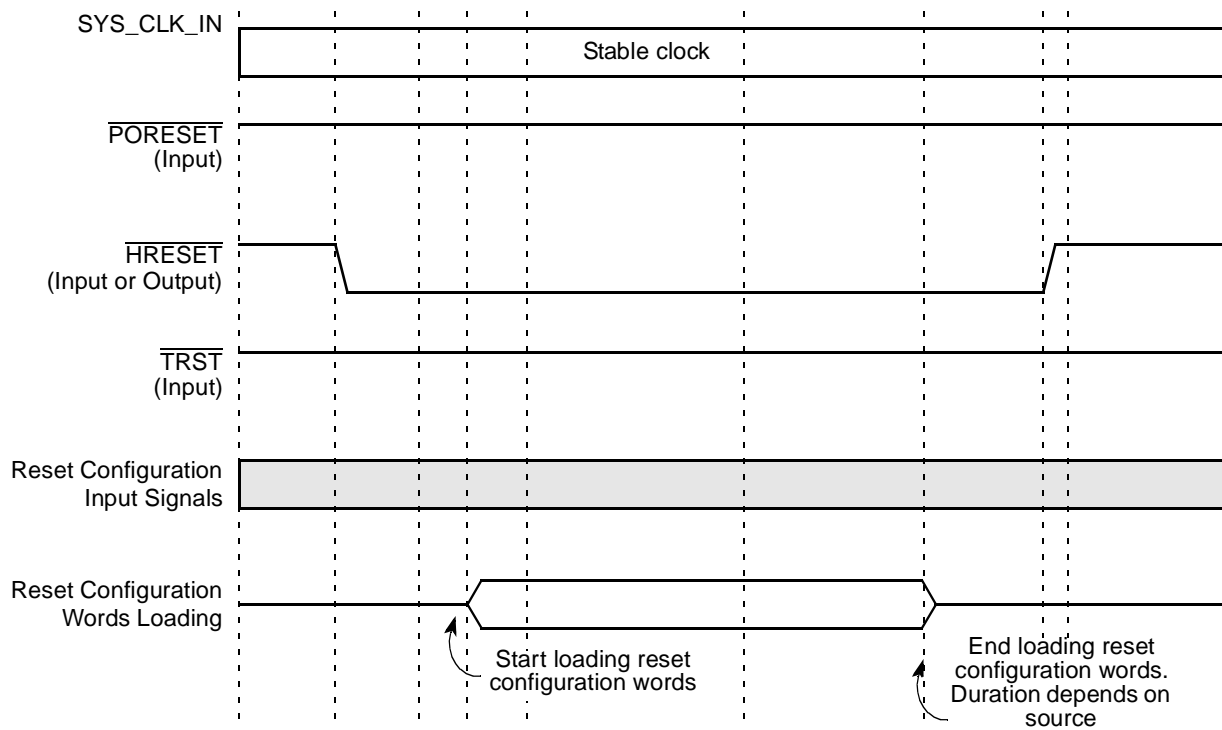


Figure 4-2. Hard Reset Flow

## 4.3 Reset Configuration

The device is initialized using two complementary methods: latching `CFG_RESET_SOURCE` and loading the reset configuration words. Initially, `CFG_RESET_SOURCE` is sampled during the assertion of the `PORESET` signal. These signals determine whether a reset configuration word is required and the device source interface from which it is loaded. According to the value on these signals, the device continues loading the reset configuration word.

### 4.3.1 Reset Configuration Signals

Reset configuration input signals are on device pins that have other functions when the device is not in reset state. These input signals are sampled into registers during the assertion of `PORESET`, after a stable clock is supplied (`SYS_CLK_IN`), and must be pulled high or low by external resistors as long as `HRESET` is asserted. While the `PORESET` or `HRESET` signal is asserted, all other signal drivers connected to these signals must be in the high-impedance state. For proper resistor values for pulling reset configuration signals high or low, see *PowerQUICC II Pro MPC8308 Hardware Specification*.

This section describes the modes configured by the reset configuration signals. Note that the reset configuration input sampled values are accessible to software through memory-mapped registers, as

described in Section 4.5.1.3, “Reset Status Register (RSR),” and Section 4.5.2.1, “System PLL Mode Register (SPMR).”

### NOTE

Implement one of the following methods to control the selection between the reset and non-reset function of these pins.

- Resistors. Use pull-up or pull-down resistors to set the desired value on the reset configuration input signals. During the power-on and hard reset sequences, these signals are inputs to the device.
- Active driving device. Use  $\overline{\text{HRESET}}$  to control the driving device. When  $\overline{\text{HRESET}}$  is asserted, drive reset configuration values on the pins; when  $\overline{\text{HRESET}}$  is negated, stop driving the reset configuration input signals.

#### 4.3.1.1 Reset Configuration Word Source

The reset configuration word source options, shown in Table 4-5, select whether the device loads a reset configuration word from NOR Flash, NAND Flash, or an I<sup>2</sup>C EEPROM or uses hard-coded default options. The value of these signals also affects the duration of power-on and hard reset sequences. In any case, the reset sequence does not exceed 1 ms.

**Table 4-5. Reset Configuration Words Source**

CFG_RESET_SOURCE[0:3]	Meaning
0000	Reset configuration word is loaded from NOR Flash
0001	Reset configuration word is loaded from NAND Flash memory (8-bit small page).
0010	Reserved
0011	Reserved
0100	Reset configuration word is loaded from an I <sup>2</sup> C EEPROM. SYS_CLK_IN is in the range of 24–66.666 MHz.
0101	Reset configuration word is loaded from NAND Flash memory (8-bit large page).
0110	Reserved
0111	Reserved
1000	Hard-coded option 0. Reset configuration word is not loaded.
1001	Hard-coded option 1. Reset configuration word is not loaded.
1010	Hard-coded option 2. Reset configuration word is not loaded.
1011	Hard-coded option 3. Reset configuration word is not loaded.
1100	Hard-coded option 4. Reset configuration word is not loaded.
1101	Reserved
1110	Reserved
1111	Reserved

### 4.3.1.2 Selecting Reset Configuration Input Signals

The example described in [Table 4-6](#) shows how the user should pull down or pull up the reset configuration input signal (CFG\_RESET\_SOURCE). The reset sequence duration is measured from the negation of  $\overline{\text{PORESET}}$  to the negation of  $\overline{\text{HRESET}}$ . Note that the duration mentioned in [Table 4-6](#) is typical, and does not represent cases in which the process of loading the reset configuration word had to be retried due to errors.

**Table 4-6. Selecting Reset Configuration Input Signals**

I <sup>2</sup> C EEPROM Configuration Words	SYS_CLK_IN Frequency	CFG_RESET_SOURCE[0:3]	Reset Sequence Duration in SYS_CLK_IN Cycles	Duration
No	33 MHz	0000 (RCW loaded from NOR Flash)	15210	456 $\mu\text{s}$
Yes	33 MHz	0100 (I <sup>2</sup> C EEPROM)	106534	196 $\mu\text{s}$

### 4.3.2 Reset Configuration Words

The reset configuration words control the clock ratios and other basic device functions, such as, boot location, eTSEC mode, and endian mode. The reset configuration words are loaded from NOR Flash, NAND Flash, or the I<sup>2</sup>C interfaces or from hard-coded values during the power-on or hard reset flows. See [Section 4.3.1, “Reset Configuration Signals,”](#) for information on the reset configuration word source. Although the configuration reset words are loaded during hard reset flows, the clocks and PLL modes are reset only when  $\overline{\text{PORESET}}$  is asserted during a power-on reset flow. See [Section 4.2.1.2, “Reset Actions.”](#) The values of fields in the reset configuration words registers (RCWLR and RCWHR) reflect only their state during the reset flow. Some of these parameters and modes can be modified by changing their values in the memory-mapped registers of other units, which does not affect RCWLR and RCWHR.

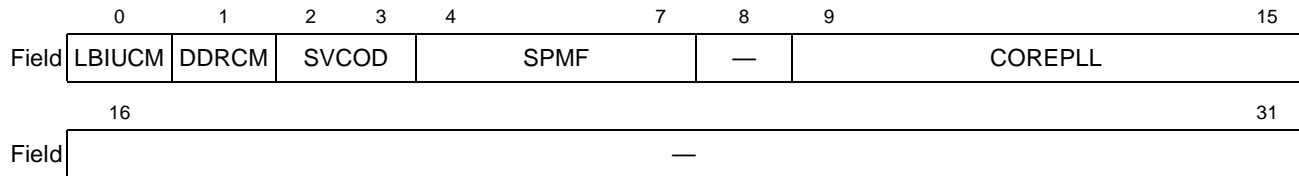
The reset configuration settings are accessible to software through the following read-only memory-mapped registers:

- Reset configuration word low register (RCWLR)
- Reset configuration word high register (RCWHR)
- Reset status register (RSR)
- System PLL mode register (SPMR)

See [Section 4.5, “Memory Map/Register Definitions.”](#)

### 4.3.2.1 Reset Configuration Word Low Register (RCWLR)

RCWLR is shown in [Figure 4-3](#). This read-only register obtains its values according to the reset configuration word low loaded during the reset flow.



**Figure 4-3. Reset Configuration Word Low Register (RCWLR)**

[Table 4-7](#) defines the RCWLR bit fields.

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

Bits	Name	Description
0	LBIUCM	Local bus memory controller clock mode. Selects the local bus controller clock ratio. The local bus memory controller operates with a frequency equal to the frequency of <i>csb_clk</i> . This bit should be cleared. LBC controller clock: <i>csb_clk</i> 0 1:1 <b>Note:</b> This bit should always be set to 0.
1	DDRCM	DDR SDRAM memory controller clock mode. Selects the DDR SDRAM memory controller clock ratio. The DDR SDRAM memory controller operates at twice the frequency of the <i>csb_clk</i> . 1 <i>csb_clk</i> ratio is 2:1 <b>Note:</b> This bit should always be set to 1.
2–3	SVCOD	System PLL VCO division. See <a href="#">Section 4.3.2.1.1, “System PLL VCO Division.”</a>
4–7	SPMF	System PLL multiplication factor. See <a href="#">Section 4.3.2.1.1, “System PLL VCO Division,”</a> for more information.
8	—	Reserved, should be cleared
9–15	COREPLL	Core PLL configuration. COREPLL sets the ratio between the e300 core clock and the internal <i>csb_clk</i> of the device. For information on the encodings for COREPLL, see <i>PowerQUICC II Pro MPC8308 Hardware Specification</i> .
16–31	—	Reserved, should be cleared.

#### 4.3.2.1.1 System PLL VCO Division

The RCWLR field SVCOD (system PLL VCO division), shown in [Table 4-8](#), establishes the internal ratio between the system PLL VCO frequency and the PLL output clock frequency. The PLL output clock frequency equals *csb\_clk* frequency if RCWLR[LBCM] and RCWLR[DDRCM] are both cleared or twice the *csb\_clk* frequency if RCWLR[LBCM] or RCWLR[DDRCM] or both of them are set.

Table 4-8 describes the setting of SVCOD bits.

**Table 4-8. System PLL VCO Division**

Reset Configuration Word Low Register (RCWLR) Bits	Field Name	Value (Binary)	VCO Division Factor
2–3	SVCOD	00	2
		01	4
		10	8
		11	Reserved

**NOTE**

For the frequency of operation for MPC8308, 00 is the only applicable value of SVCOD.

#### 4.3.2.1.2 System PLL Configuration

The system PLL ratio reset, shown in Table 4-9, establishes the clock ratio between the SYS\_CLK\_IN signal and the internal *csb\_clk* of the device. *csb\_clk* drives internal units and feeds the e300 core PLL.

**Table 4-9. System PLL Ratio**

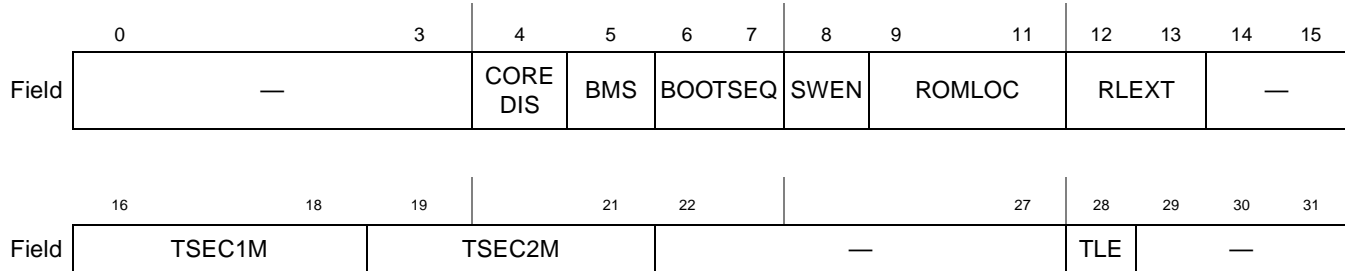
RCWLR Bits	Field Name	Value (Binary)	<i>csb_clk</i> : SYS_CLK_IN
4–7	SPMF	0000	Reserved
		0001	Reserved
		0010	2 : 1
		0011	3 : 1
		0100	4 : 1
		0101	5 : 1
		0110	6 : 1
		0111–1111	Reserved, should not be set

### 4.3.2.2 Reset Configuration Word High Register (RCWHR)

RCWHR is shown in [Figure 4-4](#). This read-only register gets its values according to the reset configuration word high loaded during the reset flow.

Offset 0x0\_0904

Access: Read/Write



**Figure 4-4. Reset Configuration Word High Register (RCWHR)**

[Table 4-10](#) defines the reset configuration word high bit fields.

**Table 4-10. Reset Configuration Word High Bit Settings**

Bits	Name	Description
0–3	—	Reserved, should be cleared
4	COREDIS	Core disable mode. Specifies the e300 core mode out of reset. If COREDIS is set, the core cannot fetch boot code until it is configured by an external master. The external master frees the core to boot by clearing the COREDIS bit in the arbiter configuration register as described in <a href="#">Section 6.2.1, “Arbiter Configuration Register (ACR).”</a> This bit must be set when the boot sequencer is enabled to initiate the device (BOOTSEQ is not 0b00). Otherwise, unpredictable behavior occurs. 0 The core can boot without waiting for configuration by an external master. 1 Core boot hold-off mode. The core is prevented from booting until it is configured by an external master.
5	BMS	Boot memory space. See <a href="#">Section 4.3.2.2.1, “Boot Memory Space (BMS),”</a> for more information.
6–7	BOOTSEQ	Boot sequencer configuration. See <a href="#">Section 4.3.2.2.2, “Boot Sequencer Configuration,”</a> for more information.
8	SWEN	Software watchdog enable. Selects whether the software watchdog is enabled to start counting down immediately when coming out of reset. The user can override this value by writing to the system watchdog control register (SWCRR[SWEN]) during system initialization. 0 Disabled 1 Enabled
9–11	ROMLOC	Boot ROM interface location. This bit combined with bit RLEXT determines where the device boots from. See <a href="#">Section 4.3.2.2.3, “Boot ROM Location,”</a> for more information.



**Table 4-10. Reset Configuration Word High Bit Settings (continued)**

Bits	Name	Description
12–13	RLEXT	Boot ROM location extension. This bit combined with bit ROMLOC determines where the device boots from. See <a href="#">Section 4.3.2.2.3, “Boot ROM Location,”</a> for more information. 00 Legacy mode—allows for booting from on-chip peripherals. For more information, see <a href="#">Table 4-13</a> . 01 NAND Flash mode—allows for booting from NAND flash devices. For more information, see <a href="#">Table 4-13</a> . 10 Reserved 11 Reserved
14–15	—	Reserved, should be cleared.
16–18	TSEC1M	TSEC1 mode See <a href="#">Section 4.3.2.2.4, “eTSEC1 Mode,”</a> for more information.
19–21	TSEC2M	TSEC2 mode. See <a href="#">Section 4.3.2.2.5, “eTSEC2 Mode,”</a> for more information.
22–27	—	Reserved, should be cleared.
28	TLE	True little-endian. See <a href="#">Section 4.3.2.2.6, “e300 Core True Little-Endian,”</a> for more information.
29–31	—	Reserved, should be cleared.

#### 4.3.2.2.1 Boot Memory Space (BMS)

BMS defines the initial value of the e300 core MSR[IP] bit, which specifies the location of the interrupt vectors (including the hard reset exception vector). The device defines the default boot ROM memory space to be 8 Mbytes at addresses 0x0000\_0000 to 0x007F\_FFFF or 0xFF80\_0000 to 0xFFFF\_FFFF. When the core comes out of reset, if it is enabled to boot, it fetches boot code from one of two addresses, 0x0000\_0100 or 0xFFFF\_0100, and exceptions are vectored to the physical addresses, 0x000n\_nnnn or 0xFFFFn\_nnnn appropriately. This bit specifies whether an interrupt vector offset is prepended with 0xFFFF or 0x000. In the description below, *n\_nnnn* is the offset of the exception vector.

The boot memory space reset configuration word field, shown in [Table 4-11](#), specifies both the device boot ROM address window and the initial e300 core boot address.

**Table 4-11. Boot Memory Space**

RCWHR Bit	Field Name	Value (Binary)	Meaning
5	BMS	0	Boot memory space is 8 Mbytes at 0x0000_0000 to 0x007F_FFFF. e300 core register MSR[IP] initial value is 0b0. The core, if enabled to boot, begins fetching boot code from address 0x0000_0100 and exceptions are vectored to the physical address of 0x000n_nnnn.
		1	Boot memory space is 8 Mbytes at 0xFF80_0000 to 0xFFFF_FFFF. e300 core register MSR[IP] initial value is 0b1. The core, if enabled to boot, begins fetching boot code from address 0xFFFF_0100 and exceptions are vectored to the physical address of 0xFFFFn_nnnn.

### 4.3.2.2.2 Boot Sequencer Configuration

The boot sequencer configuration options, shown in [Table 4-12](#), allow the boot sequencer to load configuration data from the serial ROM located on the I<sup>2</sup>C port before the host tries to configure the device. These options also specify normal or extended I<sup>2</sup>C addressing modes. See [Section 17.4.5](#), “[Boot Sequencer Mode](#).”

**Table 4-12. Boot Sequencer Configuration**

RCWHR Bits	Field Name	Value (Binary)	Meaning
6–7	BOOTSEQ	00	Boot sequencer is disabled. No I <sup>2</sup> C ROM is accessed.
		01	Normal I <sup>2</sup> C addressing mode is used. Boot sequencer is enabled and loads configuration information from a ROM on the I <sup>2</sup> C interface. A valid ROM must be present.
		10	Extended I <sup>2</sup> C addressing mode is used. Boot sequencer is enabled and loads configuration information from a ROM on the I <sup>2</sup> C interface. A valid ROM must be present.
		11	Reserved, should be cleared.

#### NOTE

When the boot sequencer is enabled, the e300 core must be prevented from fetching boot code by setting the core disable reset configuration word field (COREDIS) as described in [Section 4.3.2.2](#), “[Reset Configuration Word High Register \(RCWHR\)](#).” If the e300 core is required to proceed, the boot sequencer should enable boot vector fetch by clearing ACR[COREDIS] as described in [Section 6.2.1](#), “[Arbiter Configuration Register \(ACR\)](#).”

### 4.3.2.2.3 Boot ROM Location

The device defines the default boot ROM address range to be 8 Mbytes at addresses 0x0000\_0000 to 0x007F\_FFFF or 0xFF80\_0000 to 0xFFFF\_FFFF (selected by the BMS reset configuration word field). However, the on-chip peripheral that manages these boot ROM accesses can be selected at power up.

The boot ROM location reset configuration word field, shown in [Table 4-13](#), establishes the location of boot ROM. The exact boot ROM location table to be used is defined by the setting of RCWHR[RLEXT]

bits, as shown in [Table 4-10](#). Accesses to the boot vector and the default boot ROM region of the local address map are directed to the interface specified by this field.

**Table 4-13. Boot ROM Location**

RCWHR Bits	Field Name	Value (Binary)	Meaning	
			Legacy Mode (RLEXT = 00)	NAND Flash Mode (RLEXT = 01)
9–11	ROMLOC	000	DDR SDRAM	Reserved
		001	Reserved	Local bus NAND Flash—8-bit small page ROM
		010	Reserved	Reserved
		011	Reserved	Reserved
		100	Reserved	Reserved
		101	Local bus GPCM—8-bit ROM	Local bus NAND Flash—8-bit large page ROM
		110	Local bus GPCM—16-bit ROM	Reserved
		111	Reserved	Reserved

The local access window of the selected boot ROM interface is enabled and initialized with the proper base address and size, as described in [Section 5.1, “Local Memory Map Overview and Example.”](#)

#### 4.3.2.2.4 eTSEC1 Mode

The TSEC1 mode reset configuration word field, shown in [Table 4-14](#), selects the protocol used by the enhanced three-speed Ethernet controller interface (eTSEC1) controller.

**Table 4-14. eTSEC1 Mode Configuration**

Reset Configuration Word High Register (RCWHR) Bits	Field Name	Value (Binary)	Meaning
16–18	TSEC1M	000	The eTSEC1 controller operates in the MII protocol, using only four transmit data signals and four receive data signals.
		001	Reserved
		010	Reserved
		011	The eTSEC1 controller operates in the RGMII protocol, using four transmit data signals and four receive data signals.
		100	Reserved
		101	Reserved
		110	Reserved
		111	Reserved

#### 4.3.2.2.5 eTSEC2 Mode

The eTSEC2 mode reset configuration word field, shown in [Table 4-15](#), selects the protocol used by the eTSEC2 controller.

**Table 4-15. eTSEC2 Mode Configuration**

Reset Configuration Word High Register (RCWHR) Bits	Field Name	Value (Binary)	Meaning
19–21	TSEC2M	000	The eTSEC2 controller operates in the MII protocol, using only four transmit data signals and four receive data signals.
		001	Reserved
		010	Reserved
		011	The eTSEC2 controller operates in the RGMII protocol, using four transmit data signals and four receive data signals.
		100	Reserved
		101	Reserved
		110	Reserved
		111	Reserved

#### 4.3.2.2.6 e300 Core True Little-Endian

The true little-endian reset configuration word field, shown in [Table 4-16](#), selects whether the e300 core operates in big-endian mode or true little-endian mode at reset.

**Table 4-16. e300 Core True Little-Endian**

Reset Configuration Word High Register (RCWHR) Bit	Field Name	Value (Binary)	Meaning
28	TLE	0	Big-endian mode
		1	True little-endian mode

### 4.3.3 Loading the Reset Configuration Words

The device loads the reset configuration words from a local bus EEPROM, a local bus NAND Flash, or an I<sup>2</sup>C serial EEPROM, or uses hard-coded configuration, as selected by the reset configuration inputs described in [Section 4.3.1, “Reset Configuration Signals.”](#) The following sections describe each of these options.

#### 4.3.3.1 Loading from Local Bus

The reset configuration words are assumed to reside in an EEPROM or NOR Flash or NAND Flash device connected to  $\overline{\text{LCS0}}$  of the device local bus. Because the port size of this EEPROM is unknown, the device reads all configuration words byte-by-byte only from locations that are independent of port size.

Table 4-17 shows addresses that should be used to contain the reset configuration words. Byte addresses that do not appear in this table have no effect on the configuration of the device. The values of the bytes in Table 4-17 are always read on byte lane LD[0:7] regardless of the port size.

**Table 4-17. Local Bus Configuration EEPROM Addresses**

Reset Configuration Word	Bits [0:7] Address	Bits [8:15] Address	Bits [16:23] Address	Bits [24:31] Address
Low	0x00	0x08	0x10	0x18
High	0x20	0x28	0x30	0x38

Table 4-18 shows the data structure of the local bus device containing the reset configuration words (RCWL and RCWH).

**Table 4-18. Local Bus Reset Configuration Words Data Structure**

EEPROM Address	EEPROM Data Bits			
	[0:7]	[8:15]	[16:23]	[24:31]
0x00	RCWL[0:7]			
0x04				
0x08	RCWL[8:15]			
0x0C				
0x10	RCWL[16:23]			
0x14				
0x18	RCWL[24:31]			
0x1C				
0x20	RCWH[0:7]			
0x24				
0x28	RCWH[8:15]			
0x2C				
0x30	RCWH[16:23]			
0x34				
0x38	RCWH[24:31]			
0x3C				

#### 4.3.3.1.1 Local Bus Controller Setting

The device uses GPCM to load the reset configuration from EEPROM or NOR Flash. The device reads 64 bytes in this case. The local bus controller's registers setting is set according to Table 4-19.

The device uses FCM to load the reset configuration from NAND Flash. The device reads 512 bytes if small-page size NAND Flash is used or 2048 bytes if large-page NAND Flash is used. The local bus controller's registers setting are set according to [Table 4-19](#).

**Table 4-19. Local Bus Controller Setting When Loading RCW**

CFG_RESET_SOURCE	Meaning	BR0[PS]	BR0[MSEL]	OR0[SCY]	OR0[PGS]
0000	NOR Flash	11	000	1111	NA
0001	NAND Flash, 8 bit, small page	01	001	0010	0
0101	NAND Flash, 8 bit, large page	01	001	0010	1

### 4.3.3.2 Loading from I<sup>2</sup>C EEPROM

The device is capable of loading the reset configuration word from the I<sup>2</sup>C interface. If the device is configured to load the reset configuration word from the I<sup>2</sup>C interface according to the reset configuration input signals, the device uses the I<sup>2</sup>C unit boot sequencer in a special mode. In this mode, the I<sup>2</sup>C boot sequencer is activated while the rest of the device is still in reset state ( $\overline{\text{HRESET}}$  asserted) to load the reset configuration words from an I<sup>2</sup>C serial EEPROM.

Note that this does not prevent using the I<sup>2</sup>C boot sequencer to initiate the device in the normal functional mode after reset state has completed. The only restriction is that the first two EEPROM data structures contain dedicated reset information.

#### 4.3.3.2.1 Using the Boot Sequencer Reset Configuration

For more information on I<sup>2</sup>C interface and the boot sequencer, see [Section 17.4.5, “Boot Sequencer Mode.”](#)

#### NOTE

When reset configuration words are loaded from an I<sup>2</sup>C EEPROM, an I<sup>2</sup>C serial EEPROM of extended addressing type must be used.

If the I<sup>2</sup>C interface is used for loading the reset configuration words, the I<sup>2</sup>C module addresses the EEPROM and reads the first two data structures (after reading the preamble). Upon being read, the reset configuration words are latched inside the device and the I<sup>2</sup>C module enters its reset state until  $\overline{\text{HRESET}}$  is negated. There should be no other I<sup>2</sup>C traffic when the boot sequencer is active.

After  $\overline{\text{HRESET}}$  is negated, the functional boot sequencer, in extended I<sup>2</sup>C addressing mode, may be activated if the BOOTSEQ field of the reset configuration word high is set to 0b10.

#### 4.3.3.2.2 EEPROM Calling Address

The device uses 0b101\_0000 for the EEPROM calling address. The EEPROM to be addressed must contain the reset configuration information and be programmed to respond to this address. No additional EEPROMs are accessed by the boot sequencer in reset configuration mode.

### 4.3.3.2.3 EEPROM Data Format in Reset Configuration Mode

The I<sup>2</sup>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 0xAA\_55AA. The I<sup>2</sup>C module checks to ensure that this preamble is correctly detected before proceeding further. Following the preamble, there should be the two reset configuration words, programmed according to a particular format, as shown in [Figure 4-5](#).

The first 3 bytes hold the attributes and address offset. The addresses of the two reset configuration words must be programmed to the offset of the reset configuration word low register (RCWLR) and reset configuration word high register (RCWHR) respectively (see [Section 4.5.1.1, “Reset Configuration Word Low Register \(RCWLR\),”](#) and [Section 4.5.1.2, “Reset Configuration Word High Register \(RCWHR\)”](#)). The attributes should be programmed as follows: alternate configuration space (ACS) should be cleared (0b0), byte enables should be all ones, and continue (CONT) should be set.

After the first 3 bytes, 4 bytes of data should hold the desired value of the reset configuration word. The boot sequencer assumes that a big-endian address is stored in the EEPROM.

IMMRBAR value is prepended to the EEPROM address to generate the complete memory-mapped register’s address.

When the I<sup>2</sup>C operates in reset configuration mode, the cyclic redundancy check (CRC) is ignored, as well as any registers following the first two reset configuration words.

0	1	4	5	6	7
ACS (0)	BYTE_EN (1111)			CONT (1)	RCWLR ADDR[12–13]
RCWLR ADDR[14:21]					
RCWLR ADDR[22:29]					
Reset configuration word low [0–7]					
Reset configuration word low [8–15]					
Reset configuration word low [16–23]					
Reset configuration word low [24–31]					
ACS (0)	BYTE_EN (1111)			CONT (1)	RCWHR ADDR[12–13]
RCWHR ADDR[14–21]					
RCWHR ADDR[22–29]					
Reset configuration word high [0–7]					
Reset configuration word high [8–15]					
Reset configuration word high [16–23]					
Reset configuration word high [24–31]					

**Figure 4-5. EEPROM Data Format for Reset Configuration Words Preload Command**

[Figure 4-6](#) shows an example of the CRC and EEPROM contents, including the preamble, reset configuration words and additional initialization data. In this example, it is assumed that the EEPROM

contains information additional to the reset configuration words, which should be loaded in the functional state after the device completes its reset flow.

0	1	2	3	4	5	6	7	Preamble	
1	0	1	0	1	0	1	0		
0	1	0	1	0	1	0	1		
1	0	1	0	1	0	1	0		
0	1	1	1	1	1	RCWLR ADDR[12-13]		Reset configuration word low preload command	
RCWLR ADDR[14-21]									
RCWLR ADDR[22-29]									
Reset configuration word low [0-7]									
Reset configuration word low [8-15]									
Reset configuration word low [16-23]									
Reset configuration word low [24-31]									
0	1	1	1	1	1	RCWHR ADDR[12-13]		Reset configuration word high preload command	
RCWHR ADDR[14-21]									
RCWHR ADDR[22-29]									
Reset configuration word high [0-7]									
Reset configuration word high [8-15]									
Reset configuration word high [16-23]									
Reset configuration word high [24-31]									
*									
ACS	BYTE_EN				1	ADDR[12-13]			Last configuration preload command
ADDR[14-21]									
ADDR[22-29]									
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		
CRC[0-7]							Cyclic redundancy check		
CRC[8-15]									
CRC[16-23]									
CRC[24-31]									

Figure 4-6. EEPROM Contents



#### 4.3.3.2.4 Reset Configuration Load Fail

Failure of reset configuration load by the I<sup>2</sup>C boot sequencer can be caused by an incorrect EEPROM data structure or I<sup>2</sup>C bus problem. If a reset configuration load failure occurs, due to preamble fail or any other I<sup>2</sup>C bus error detection, the device continuously attempts to reload the hard reset configuration words from the I<sup>2</sup>C bus. The device does not negate  $\overline{\text{HRESET}}$  and remains in hard reset state until the RCWs are successfully loaded or the PORESET flow is restarted.

#### 4.3.3.3 Default Reset Configuration Words

If the device is configured not to load the reset configuration words from NOR Flash, NAND Flash, or an I<sup>2</sup>C EEPROM, it can also be initialized with one of five hard-coded default options, selected by the reset configuration input signals, CFG\_RESET\_SOURCE[0:3].

The reset configuration words are driven internally with the values shown in [Table 4-20](#), [Table 4-21](#), and [Table 4-22](#).

**Table 4-20. RCW Values Corresponding to Hard Coded Options**

CFG_RESET_SOURCE	RCWLR [0:31]	RCWHR [0:31]
4'b1000	32'h4504_0000	32'h0460_0000
4'b1001	32'h4404_0000	32'h0460_6C00
4'b1010	32'h4405_0000	32'h0C60_6C00
4'b1011	32'h4406_0000	32'h0460_0000
4'b1100	32'h4406_0000	32'h0460_6C00

**Table 4-21. Hard Coded Reset Configuration Word Low Fields Values**

RCWLR								
Bit	0	1	2–3	4–7	8	9–10	11–15	16–31
Definition	LBCM	DDRCM	SVCOD	SPMF	Reserve	CVCOD	CPMF	Reserve
RCWLR = 32'h4423_0000 (sysclk = 31.25)								
Value	0	1	00	0100	0	01	00011	0
Comment	csb_clk	csb_clkx2	sppll_out x2	4:1	—	cppll_out x4	1.5:1	—
Frequency	125	250	500	125	—	750	187.5	—
RCWLR = 32'h4404_0000 (sysclk = 33.33)								
Value	0	1	00	0100	0	00	00100	0
Comment	csb_clk	csb_clkx2	sppll_out x2	4:1	—	cppll_out x2	2:1	—
Frequency	133	266	533	133	—	533	266	—

Table 4-22 defines the hard-coded reset configuration word high fields values. These values select hard-coded reset configuration words options, as described in Section 4.3.1.1, “Reset Configuration Word Source.”

**Table 4-22. Hard-Coded Reset Configuration Word High Field Values**

Bits	Name	Field Values when CFG_RESET_SOURCE[0-3] = 1000-1100					Meaning
		1000	1001	1010	1011	1100	
0-3	Reserved	0000					—
4	COREDIS	0	0	1	0	0	e300 core is disabled (boot hold-off)
5	BMS	1					Boot memory space is 0xFF80_0000–0xFFFF_FFFF. MSR[IP] initial value is 0b1.
6-7	BOOTSEQ	00					Boot sequencer is disabled
8	SWEN	0					Software watchdog disabled
9-11	ROMLOC	110					Boot ROM interface location
12-13	RLEXT	00					Legacy mode
14-15	Reserved	00					—
16-18	TSEC1M	000	011	011	000	011	000 = MII mode 011 = RGMII mode
19-21	TSEC2M	000	011	011	000	011	
22-27	Reserved	000000					—
28	TLE	0					Big-endian mode
29-31	Reserved	000					—

## 4.4 Clocking

The following external clock sources are utilized on MPC8308:

- System clock (SYS\_CLK\_IN)
- Ethernet Clock (TSEC<sub>n</sub>\_RX\_CLK/TSEC<sub>n</sub>\_TX\_CLK/TSEC<sub>n</sub>\_GTX\_CLK<sub>125</sub> for eTSEC1 and eTSEC2)
- USB clock (USB\_CLK\_IN for ULPI)
- Real-time clock (RTC\_PIT\_CLK)
- SerDes PHY clock

For more information, see Chapter 15, “SerDes PHY.”

All clock inputs can be supplied using an external canned oscillator, a clock generation chip, or some other source that provides a standard CMOS square wave input.

Figure 4-7 shows the internal distribution of clocks within the device.

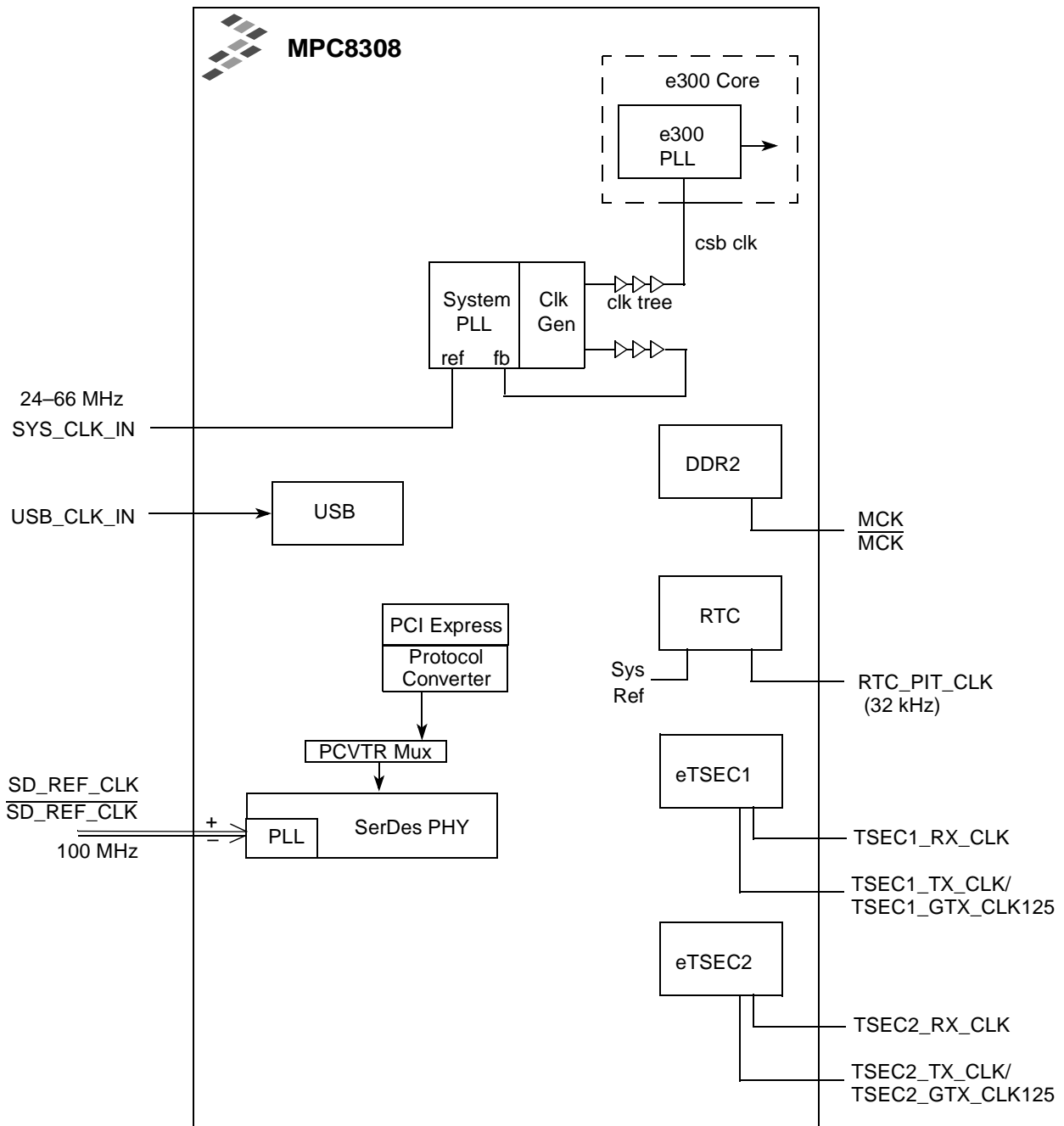


Figure 4-7. Clock Subsystem Block Diagram

### 4.4.1 System Clock Domains

As shown in [Figure 4-7](#), the primary clock input (SYS\_CLK\_IN) frequency is multiplied up by the system phase-locked loop (PLL) and the clock unit to create three major clock domains:

- The coherent system bus clock (*csb\_clk*)
- The internal clock for the DDR controller (*ddr\_clk*)
- The internal clock for the local bus interface unit (*lbc\_clk*)

The *csb\_clk* frequency is derived as follows:

$$csb\_clk = [SYS\_CLK\_IN] \times SPMF$$

The *csb\_clk* serves as the clock input to the e300 core. A second PLL inside the core multiplies up the *csb\_clk* frequency to create the internal clock for the core (*core\_clk*). The system and core PLL multipliers are selected by the SPMF and COREPLL fields in the reset configuration word low (RCWL), which is loaded at power-on reset or by one of the hard-coded reset options. See [Section 4.3, “Reset Configuration.”](#)

The DDR SDRAM memory controller operates with a frequency equal to twice the frequency of *csb\_clk*. Note that *ddr\_clk* is not the external memory bus frequency; *ddr\_clk* passes through the DDR clock divider ( $\div 2$ ) to create the differential DDR memory bus clock outputs (MCK and  $\overline{MCK}$ ). However, the data rate is the same frequency as *ddr\_clk*.

The local bus memory controller operates with a frequency equal to the frequency of *csb\_clk*. Note that *lbc\_clk* is not the external local bus frequency; *lbc\_clk* passes through the LBC clock divider to create the external local bus clock output (LCLK). The LBC clock divider ratio is controlled by LCCR[CLKDIV]. See [Section 10.1.3.1, “eLBC Bus Clock and Clock Ratios,”](#) for more information.

In addition, some of the internal units may be required to be shut off or operate at lower frequency than the *csb\_clk* frequency. These units have a default clock ratio that can be configured by a memory-mapped register after the device comes out of reset. [Table 4-23](#) specifies which units have a configurable clock frequency. For more information, see [Section 4.5.2.3, “System Clock Control Register \(SCCR\).”](#)

**Table 4-23. Configurable Clock Units**

Unit	Default Frequency	Options
eTSEC1 and eTSEC2	<i>csb_clk</i>	Off, <i>csb_clk</i> , <i>csb_clk</i> /2, <i>csb_clk</i> /3
I <sup>2</sup> C	<i>csb_clk</i>	Off, <i>csb_clk</i> , <i>csb_clk</i> /2, <i>csb_clk</i> /3
USB DR	<i>csb_clk</i>	Off, <i>csb_clk</i> , <i>csb_clk</i> /2, <i>csb_clk</i> /3
DMA	<i>csb_clk</i>	Off, <i>csb_clk</i> , <i>csb_clk</i> /2, <i>csb_clk</i> /3
PCIEXP	<i>csb_clk</i>	Off, <i>csb_clk</i> , <i>csb_clk</i> /2, <i>csb_clk</i> /3
eSDHC	<i>csb_clk</i>	Off, <i>csb_clk</i> , <i>csb_clk</i> /2, <i>csb_clk</i> /3

#### NOTE

The clock ratios of these units must be set before they are accessed.

eTSEC1 and eTSEC2 share the same clock control, and therefore the same clock ratio. They can be independently switched off.

## 4.4.2 USB Clocking

The clock signal (USB\_CLK\_IN) is given from an external source. For more details refer to [Chapter 13, “Universal Serial Bus Interface.”](#)

## 4.4.3 Ethernet Clocking

When running in RGMII mode, the reference clocks are TSEC<sub>n</sub>\_GTX\_CLK125 and TSEC<sub>n</sub>\_RX\_CLK input on the eTSEC1 and eTSEC2 interface. This can either be a 2.5- or 3.3-V signal.

When running in the MII mode, the reference clocks are TSEC<sub>n</sub>\_TX\_CLK and TSEC<sub>n</sub>\_RX\_CLK input on the eTSEC1 and eTSEC2 interface. This can be either a 2.5- or 3.3-V signal.

For more information, see [Chapter 16, “Enhanced Three-Speed Ethernet Controllers.”](#)

## 4.5 Memory Map/Register Definitions

This section presents memory maps and register descriptions for both reset and clocking.

### 4.5.1 Reset Configuration Register Descriptions

The reset configuration and status registers are shown in [Table 4-24](#).

**Table 4-24. Reset Configuration and Status Registers Memory Map**

Address	Register	Access	Reset	Section/Page
<b>Reset Configuration—Block Base Address 0x0_0900</b>				
0x0_0900	Reset configuration word low register (RCWLR)	R	0x0000_0000	<a href="#">4.5.1.1/4-25</a>
0x0_0904	Reset configuration word high register (RCWHR)	R	0x0000_0000	<a href="#">4.5.1.2/4-26</a>
0x0_0908– 0x0_090C	Reserved, should be cleared	—	—	—
0x0_0910	Reset status register (RSR)	R/W	0x0000_0000	<a href="#">4.5.1.3/4-26</a>
0x0_0914	Reset mode register (RMR)	R/W	0x0000_0000	<a href="#">4.5.1.4/4-27</a>
0x0_0918	Reset protection register (RPR)	R/W	0x0000_0000	<a href="#">4.5.1.5/4-28</a>
0x0_091C	Reset control register (RCR)	R/W	0x0000_0000	<a href="#">4.5.1.6/4-28</a>
0x0_0920	Reset control enable register (RCER)	R/W	0x0000_0000	<a href="#">4.5.1.7/4-29</a>
0x0_0924	Reserved.	—	—	—
0x0_0928– 0x0_09FC	Reserved, should be cleared	—	—	—

#### 4.5.1.1 Reset Configuration Word Low Register (RCWLR)

The reset configuration word low register (RCWLR) is shown in [Figure 4-3](#) and described in [Section 4.3.2.1, “Reset Configuration Word Low Register \(RCWLR\).”](#)

### 4.5.1.2 Reset Configuration Word High Register (RCWHR)

The reset configuration word high register (RCWHR) is shown in [Figure 4-4](#) and described in [Section 4.3.2.2, “Reset Configuration Word High Register \(RCWHR\).”](#)

### 4.5.1.3 Reset Status Register (RSR)

RSR, shown in [Figure 4-8](#), captures various reset events in the device. The RSR accumulates reset events. For example, because software watchdog expiration results in a hard reset, SWRS and HRS are all set after a software watchdog reset. This register returns to its reset value only when power-on reset occurs.

Address 0x0\_0910

Access: User read/write

	0	3	4							14	15		
R	RSTSRC			—						BSF			
W	n <sup>1</sup>			0						0			
Reset	0			0						0			
	16	18	19	20	23	24	26	27	28	29	30	31	
R	—		SWHR	—			—		CSHR	SWRS	BMRS	—	HRS
W	0		0	0			0		0	0	0	0	
Reset	0		0	0			0		0	0	0	0	

<sup>1</sup> The reset value of this field is determined according to the reset configuration input signals CFG\_RESET\_SOURCE[0:2] sampled during the reset flow.

**Figure 4-8. Reset Status Register (RSR)**

[Table 4-25](#) defines the reset status register bit fields.

**Table 4-25. Reset Status Register Field Descriptions**

Bits	Name	Description
0–3	RSTSRC	Reset configuration word source. Reflects the value of CFG_RESET_SOURCE input signal during the reset flow. See <a href="#">Section 4.3.1.1, “Reset Configuration Word Source.”</a> Changing this field has no effect.
4–14	—	Reserved, should be cleared.
15	BSF	Boot sequencer fail. If set, indicates that the I <sup>2</sup> C boot sequencer has failed while loading the reset configuration words. Cleared by writing a 1 to it (writing zero has no effect).
16–18	—	Reserved, should be cleared.
19	SWHR	Software hard reset. If set, indicates a software hard reset. SWHR is cleared by writing a 1 to it (writing zero has no effect).
20–23	—	Reserved
24–26	—	Reserved, should be cleared.
27	CSHR	Check stop reset status. When the core enters a checkstop state and the checkstop reset is enabled by the RMR[CSRE], CSRS is set and it remains set until software clears it. CSRS is cleared by writing a 1 to it (writing zero has no effect). 0 No enabled check stop reset event. 1 Enabled check stop reset event.

**Table 4-25. Reset Status Register Field Descriptions (continued)**

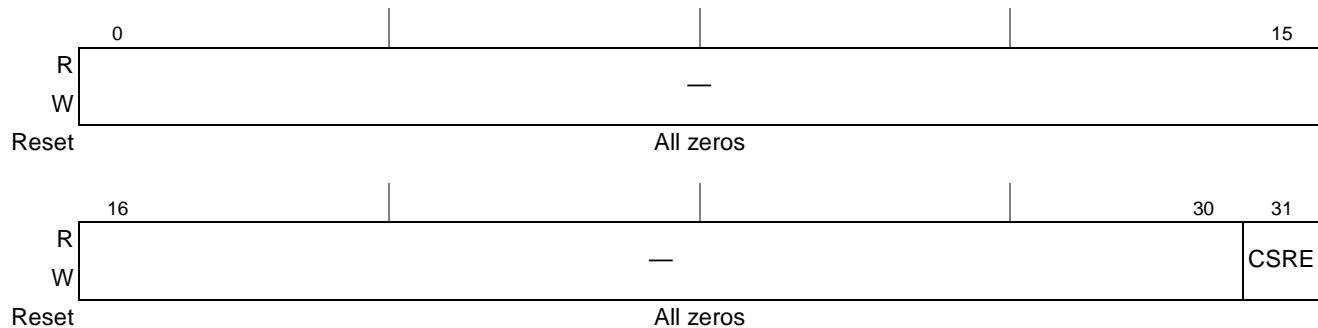
Bits	Name	Description
28	SWRS	Software watchdog reset status. When a software watchdog expire event (which causes a reset) is detected, SWRS is set and remains that way until the software clears it. SWRS is cleared by writing a 1 to it (writing zero has no effect). 0 No software watchdog reset event. 1 Software watchdog reset event.
29	BMRS	Bus monitor reset status. When a bus monitor expire event (which causes a reset) is detected, BMRS is set and remains set until the software clears it. BMRS can be cleared by writing a 1 to it (writing zero has no effect). 0 No bus monitor reset event. 1 Bus monitor reset event.
30	—	Reserved
31	HRS	Hard reset status. When an external or internal hard reset event is detected, HRS is set and remains set until software clears it. HRS is cleared by writing a 1 (writing zero has no effect). 0 No hard reset event. 1 Hard reset event.

#### 4.5.1.4 Reset Mode Register (RMR)

RMR, shown in [Figure 4-9](#), enables a hard reset sequence on the device when the e300 core enters checkstop state.

Address 0x0\_0914

Access: User read/write

**Figure 4-9. Reset Mode Register (RMR)**

[Table 4-26](#) describes the RMR fields.

**Table 4-26. RMR Field Descriptions**

Bits	Name	Function
0–30	—	Reserved, should be cleared.
31	CSRE	Checkstop reset enable. The core can enter checkstop mode as the result of several exception conditions. Setting CSRE configures the device to perform a hard reset sequence when the core enters checkstop state. 0 Reset not generated when core enters checkstop state. 1 Reset generated when core enters checkstop state.

### 4.5.1.5 Reset Protection Register (RPR)

RPR, shown in Figure 4-10, prevents unintended software reset requests caused by writes to the reset control register (RCR). To disable a write to the reset control register (RCR), the user should write a 1 to RCER[CRE].

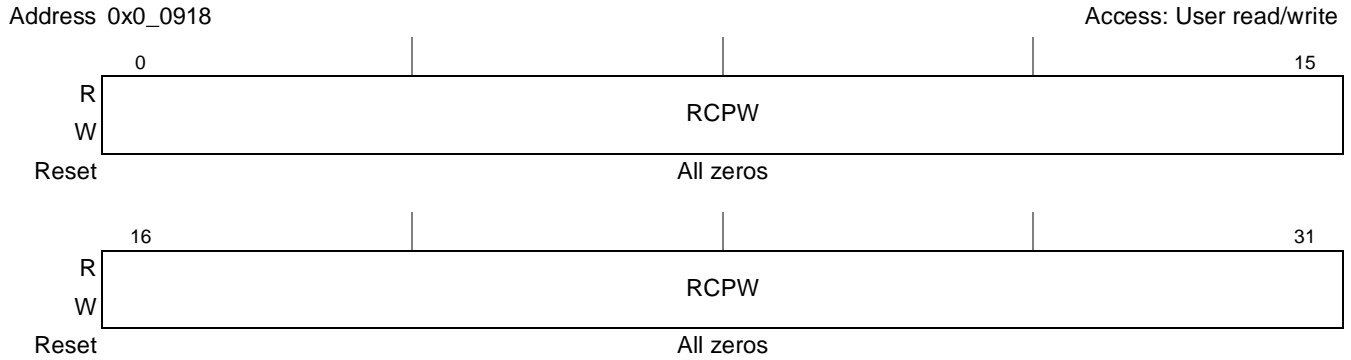


Figure 4-10. Reset Protection Register (RPR)

Table 4-27 defines the bit fields of RPR.

Table 4-27. RPR Bit Descriptions

Bits	Name	Description
0–31	RCPW	Reset control protection word. Prevents unintended software reset requests because of a write to the RCR. The user should write the value 0x5253_5445 (RSTE in ASCII) to enable. Enable indication appears in the reset control enable register (RCER[CRE]). Reading this register always returns all zeros.

### 4.5.1.6 Reset Control Register (RCR)

RCR, shown in Figure 4-11, can be used by software to initiate a hard reset sequence. To allow writing to this register, the user must enable it by writing the value 0x5253\_5445 to the RPR.

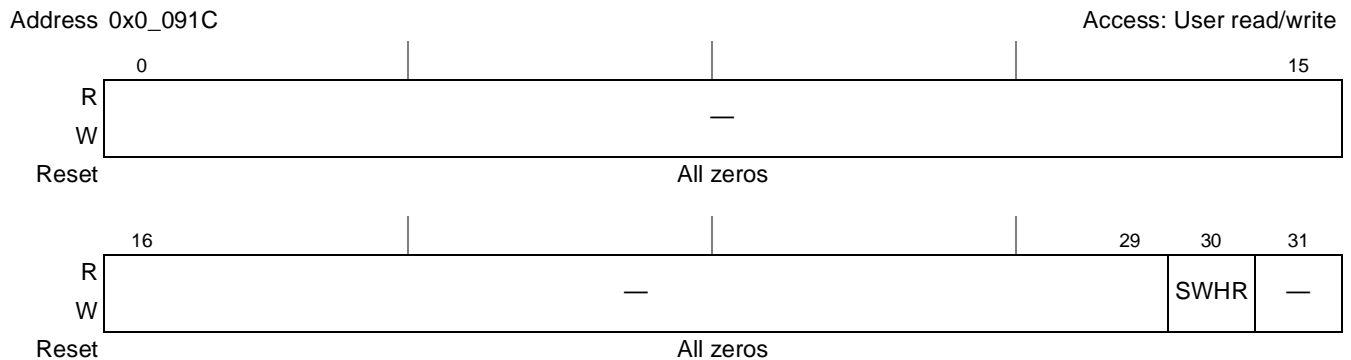


Figure 4-11. Reset Control Register (RCR)



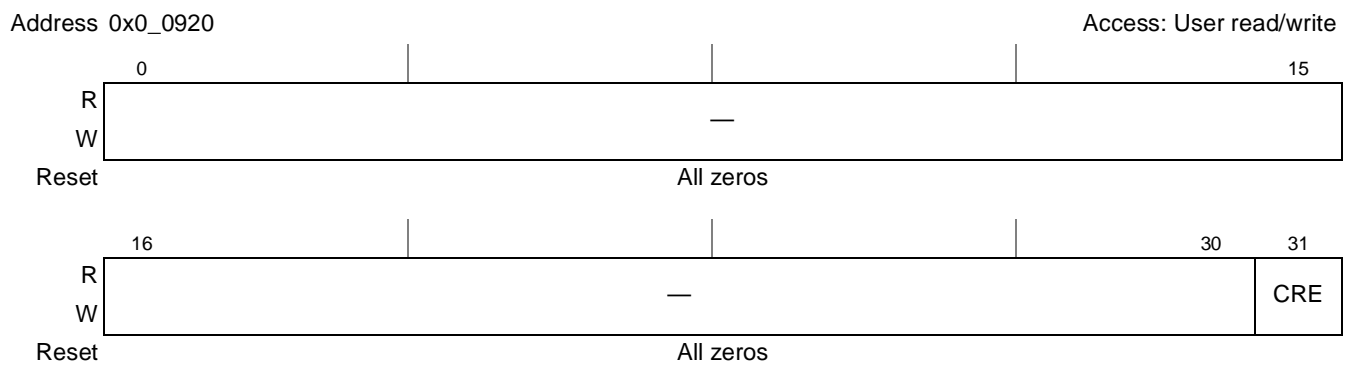
Table 4-28 defines the bit fields of RCR.

**Table 4-28. RCR Bit Settings**

Bits	Name	Description
0–29	—	Reserved, should be cleared.
30	SWHR	Software hard reset. Setting this bit causes the device to begin a hard reset flow. This bit returns to its reset state during the reset sequence, so reading it always returns all zeros.
31	—	Reserved.

#### 4.5.1.7 Reset Control Enable Register (RCER)

RCER, shown in Figure 4-12, indicates by the CRE field that the RPR is accessed with a value that enables RCR.



**Figure 4-12. Reset Control Enable Register (RCER)**

Table 4-29 defines the bit fields of RCER.

**Table 4-29. RCER Bit Settings**

Bits	Name	Description
0–30	—	Reserved, should be cleared.
31	CRE	Control register enabled. When set, indicates that the RPR was accessed with a value that enables the RCR. Writing 1 to this bit disables the RCR and clears this bit. Writing zero has no effect.

## 4.5.2 Clock Configuration Registers

The clock configuration and status registers are shown in Table 4-30.

**Table 4-30. Clock Configuration Registers Memory Map**

Address	Register	Access	Reset	Section/Page
<b>Reset Configuration—Block Base Address 0x0_0A00</b>				
0x0_0A00	System PLL mode register (SPMR)	R	0xnnnn_nnnn	4.5.2.1/4-30
0x0_0A04	Output clock control register (OCCR)	R/W	0x0000_E080	4.5.2.2/4-31

Table 4-30. Clock Configuration Registers Memory Map (continued)

Address	Register	Access	Reset	Section/Page
0x0_0A08	System clock control register (SCCR)	R/W	0x5550_0010	<a href="#">4.5.2.3/4-32</a>
0x0_0A0C– 0x0_0AFC	Reserved, should be cleared	—	—	—

### 4.5.2.1 System PLL Mode Register (SPMR)

SPMR is shown in [Figure 4-13](#). It obtains its values according to the reset configuration input signal and the reset configuration word low loaded during the reset flow. Note that this register is updated only during a power-on reset sequence and not by a hard reset sequence. It may hold values different than those in the RCWLR after a hard reset sequence.

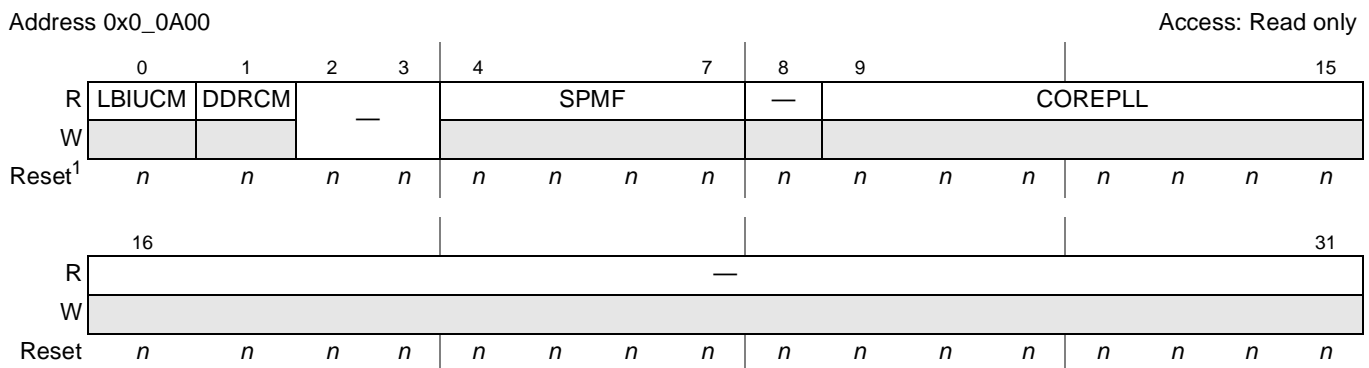


Figure 4-13. System PLL Mode Register

<sup>1</sup> See [Table 4-31](#) for reset values.

[Table 4-31](#) defines the system PLL mode register bit fields.

Table 4-31. System PLL Mode Register Bit Settings

Bits	Name	Meaning	Description
0	LBIUCM	Local bus memory controller clock mode	<a href="#">Section 4.3.2.1, “Reset Configuration Word Low Register (RCWLR)”</a>
1	DDRCM	DDR SDRAM memory controller clock mode	<a href="#">Section 4.3.2.1, “Reset Configuration Word Low Register (RCWLR)”</a>
2–3	—	Reserved, should be cleared	—
4–7	SPMF	System PLL multiplication factor	<a href="#">Section 4.3.2.1.2, “System PLL Configuration”</a>
8	—	Reserved	—
9–15	COREPLL	Core PLL configuration	For more information, see <i>PowerQUICC II Pro MPC8308 Hardware Specification</i> .
16–31	—	Reserved, should be cleared	—

### 4.5.2.2 Output Clock Control Register (OCCR)

The OCCR shown in Figure 4-14, controls the device output clocks. It is possible to control some output clock modes by writing to this memory-mapped register as described below.

Address 0x0\_0A04

Access: Read/Write

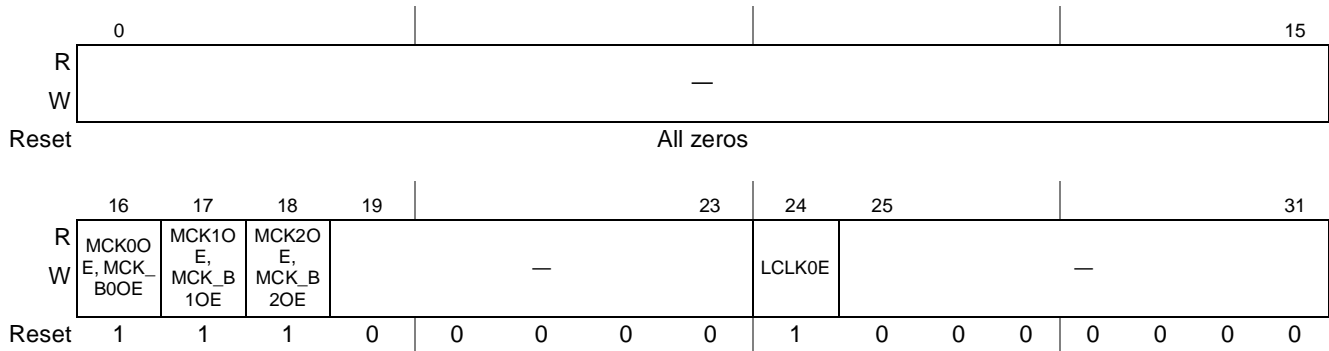


Figure 4-14. Output Clock Control Register (OCCR)

Table 4-32 defines the bit fields of OCCR.

Table 4-32. OCCR Bit Settings

Bits	Name	Description
0–15	—	Reserved, should be cleared
16	MCK0OE, MCK_B0OE	Enable/Disable MCK[0] pins clock out 0 Disable MCK[0] and $\overline{\text{MCK}}[0]$ 1 Enable MCK[0] and $\overline{\text{MCK}}[0]$
17	MCK1E, MCK_B1OE	Enable/Disable MCK[1] pins clock out 0 Disable MCK[1] and $\overline{\text{MCK}}[1]$ 1 Enable MCK[1] and $\overline{\text{MCK}}[1]$
18	MCK2E, MCK_B2OE	Enable/Disable MCK[2] pins clock out 0 Disable MCK[2] and $\overline{\text{MCK}}[2]$ 1 Enable MCK[2] and $\overline{\text{MCK}}[2]$
19–23	—	Reserved, should be cleared
24	LCLK0E	Enable/Disable LCLK[0] pin clock out 0 Disable LCLK[0] 1 Enable LCLK[0]
25–31	—	Reserved, should be cleared

### 4.5.2.3 System Clock Control Register (SCCR)

The system clock control register (SCCR), shown in Figure 4-15, controls device units that have a configurable clock ratio.

#### NOTE

The SCCR is not meant for dynamic On/Off of the clock to the module. This can be only disabled once after reset. To use the module again, a power-on reset cycle has to take place.

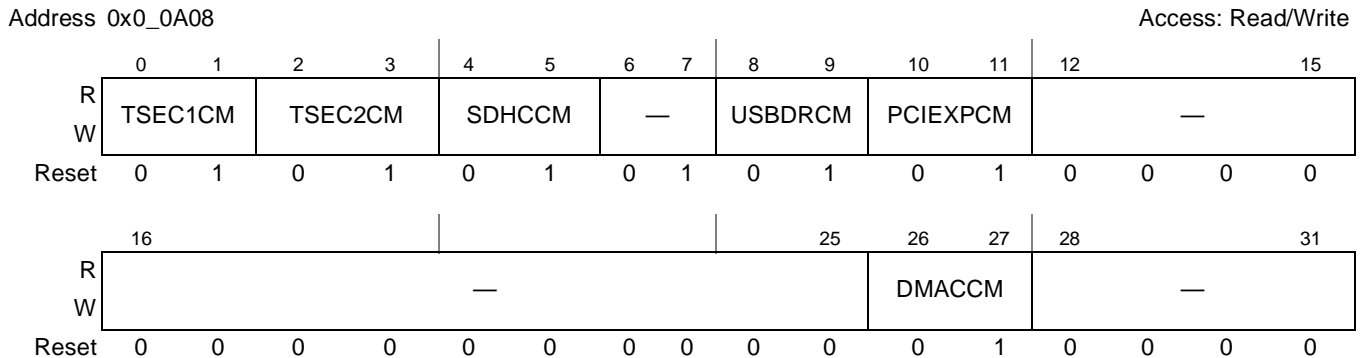


Figure 4-15. System Clock Control Register (SCCR)

Table 4-33 defines the bit fields of SCCR.

Table 4-33. SCCR Bit Descriptions

Bits	Name	Description
0–1	TSEC1CM	TSEC1 clock mode. 00 TSEC1 clock is disabled. 01 TSEC1 clock/ <i>csb_clk</i> ratio is 1:1. 10 TSEC1 clock/ <i>csb_clk</i> ratio is 1:2 ( <i>csb_clk</i> has higher frequency than TSEC1). 11 TSEC1 clock/ <i>csb_clk</i> ratio is 1:3 ( <i>csb_clk</i> has higher frequency than TSEC1).
2–3	TSEC2CM	TSEC2 clock mode. 00 TSEC2 clock is disabled. 01 TSEC2 clock/ <i>csb_clk</i> ratio is 1:1. 10 TSEC2 clock/ <i>csb_clk</i> ratio is 1:2 ( <i>csb_clk</i> has higher frequency than TSEC2). 11 TSEC2 clock/ <i>csb_clk</i> ratio is 1:3 ( <i>csb_clk</i> has higher frequency than TSEC2).
4–5	SDHCCM	SDHC clock mode 00 SDHC core clock is disabled. 01 SDHC core clock/ <i>csb_clk</i> ratio is 1:1. 10 SDHC core clock/ <i>csb_clk</i> ratio is 1:2. 11 SDHC core clock/ <i>csb_clk</i> ratio is 1:3.
6–7	—	Reserved, should be set to 0 1
8–9	USBDRCM	USB DR clock mode. 00 USB DR clock is disabled. 01 USB DR clock/ <i>csb_clk</i> ratio is 1:1. 10 USB DR clock/ <i>csb_clk</i> ratio is 1:2 ( <i>csb_clk</i> has higher frequency than the USB DR). 11 USB DR clock/ <i>csb_clk</i> ratio is 1:3 ( <i>csb_clk</i> has higher frequency than the USB DR).

Table 4-33. SCCR Bit Descriptions (continued)

Bits	Name	Description
10–11	PCIEXPCM	PCIEXP1 clock mode. Define the clock mode for the PCI Express 1 controller. 00 PCIEXP1 clock is disabled. 01 PCIEXP1 clock is enabled.
12–25	—	Reserved
26–27	DMACCM	DMACCM 00 DMAC core clock is disabled. 01 DMAC core clock/ <i>csb_clk</i> ratio is 1:1. 10 DMAC core clock/ <i>csb_clk</i> ratio is 1:2. 11 DMAC core clock/ <i>csb_clk</i> ratio is 1:3.
28–31	—	Reserved



# Chapter 5

## System Configuration

This chapter describes several functions that control the local access windows, system configuration, protection, and general utilities. These functions are discussed in the following sections:

- [Section 5.1, “Local Memory Map Overview and Example”](#)
- [Section 5.2, “System Configuration”](#)
- [Section 5.3, “Software Watchdog Timer \(WDT\)”](#)
- [Section 5.4, “Real Time Clock \(RTC\) Module”](#)
- [Section 5.5, “Periodic Interval Timer \(PIT\)”](#)
- [Section 5.6, “General-Purpose Timers \(GTM\)”](#)
- [Section 5.7, “Power Management Control \(PMC\)”](#)

### 5.1 Local Memory Map Overview and Example

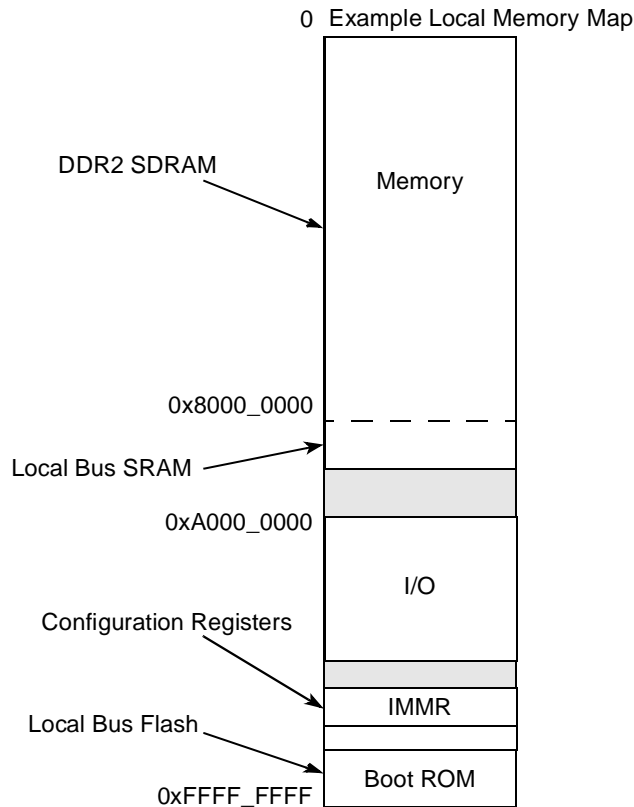
The device provides a flexible local memory map. The local memory map refers to the 32-bit address space seen by the processor as it accesses memory and I/O space. Internal DMA engines also see this same local memory map. All memory accessed by the DDR SDRAM and local bus memory controllers exists in this memory map, as do all memory-mapped configuration, control, and status registers.

The local memory map is defined by a set of eight local access windows. Each of these windows maps a region of memory to a particular target interface, such as the DDR SDRAM 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 2 Gbytes. Each local access window is assigned to a specific target interface as specified in [Table 5-1](#).

**Table 5-1. Local Access Windows Target Interface**

Window Number	Target Interface	Comments
0	Configuration registers (IMMR)	Fixed 1-Mbyte window size
1	Local bus	—
2	Local bus	—
3	Local bus	—
4	Local bus	—
5	DDR2 SDRAM	—
6	DDR2 SDRAM	—
7	PCI Express	—

Figure 5-1 shows an example memory map.



**Figure 5-1. Local Memory Map Example**

Table 5-2 shows an example of local access window settings.

**Table 5-2. Local Access Windows Example**

Window	Base Address	Size	Target Interface
5	0x0000_0000	2 Gbytes	DDR2 SDRAM
2	0x8000_0000	1 Mbyte	Local bus
7	0x8100_0000	1 Mbyte	PCI Express
3	0xC000_0000	256 Mbytes	Local bus
0	0xFF40_0000	1 Mbyte	Configuration registers (IMMR)
1	0xFF80_0000	8 Mbytes	Local bus boot ROM Flash
6,4	Unused		

In this example, the local access window of the boot ROM is defined as window number 1, on a local bus device, in the highest 8 Mbytes of memory as set by the reset configuration word high during the reset sequence (see [Section 4.3.2.2.3, “Boot ROM Location”](#)) and [Section 5.1.4.3.1, “LBLAWBAR0\[BASE\\_ADDR\] Reset Value.”](#) The local access window, which describes the range of memory used for memory-mapped registers (IMMR), is a fixed 1-Mbyte space pointed to by the IMMRBAR register using its default value (0xFF40\_0000). See [Section 5.1.4.1, “Internal Memory Map Registers Base Address Register \(IMMRBAR\).”](#)



### 5.1.1 Address Translation and Mapping

In addition to any address translation performed by the e300 core MMU, three distinct types of translation and mapping operations are performed on transactions at the integrated device level. These are as follows:

- Mapping a local address to a target interface
- Translating the local 32-bit address to an external address space
- Translating external addresses to the local 32-bit address space

The local access windows perform target mapping for transactions within the local address space. The local access windows do not perform any address translation.

Outbound windows perform the mapping from the local 32-bit address space to the address space of the PCI Express interface, which may be much larger than the local space.

Inbound windows perform address translation from the external address spaces of the PCI Express interface to the local address space.

The target mappings created by an inbound window must be consistent with those of the local access windows. That is, if an inbound window maps a transaction to a given local address, a valid local access window for that address must be set independently.

All of the configuration registers that define mapping of local access windows follow the same register format. [Table 5-3](#) summarizes the general format of these window definitions.

**Table 5-3. Format of Window Definitions**

Register	Function
Base address	High-order address bits defining location of the window in the initial address space
Window size/attributes	Window enable, window size <sup>1</sup>

<sup>1</sup> An exception is the IMMR window, which is always enabled and has a fixed 1-Mbyte size.

Windows must be a power-of-two size. To perform a 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 within a window, the transaction is directed to the appropriate target.

### 5.1.2 Window into Configuration Space

The internal memory map registers' base address register (IMMRBAR) defines a window that is used to access all memory-mapped configuration, control, and status registers, referred to as internal memory map registers or IMMR. This window is always enabled with a fixed size of 1 Mbyte, and no other attributes are attached so there is no associated size/attribute register. This window always takes precedence over all local access windows. The IMMRBAR always come out of reset with a default base address value of

0xFF40\_0000, and this base address value can be modified by writing to this register. For more information, see [Section 5.1.4.1, “Internal Memory Map Registers Base Address Register \(IMMRBAR\).”](#)

### NOTE

Although it is legal to use the 3-Mbyte space consecutive to the 1 Mbyte of the IMMR (for example, if IMMRBAR is 0xFF40\_0000, the 3-Mbyte address space consecutive to it is 0xFF50\_0000–0xFF7F\_FFFF), it is not recommended. This space may be used in future derivatives of the device that require a larger internal memory space.

## 5.1.3 Local Access Windows

As demonstrated in the address map overview in [Section 5.1, “Local Memory Map Overview and Example,”](#) local access windows associate a range of the local 32-bit address space with a particular target interface. This allows the internal interconnections of the device 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 and define its size, while the window number specifies the target interface.

With the exception of configuration space (mapped by IMMRBAR), all addresses used by the system must be mapped by a local access window. This includes addresses that are mapped by PCI Express inbound windows.

The local access window registers exist as part of the local access block in the system configuration registers. See [Section 5.2.2, “System Configuration 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.

### 5.1.3.1 Local Access Register Memory Map

[Table 5-4](#) shows the memory map for the local access registers.

**Table 5-4. Local Access Register Memory Map**

Local Access—Block Base Address 0x0_0000				
Local Memory Offset (Hex)	Register	Access	Reset	Section/Page
0x0_0000	Internal memory map base address register (IMMRBAR)	R/W	0xFF40_0000	<a href="#">5.1.4.1/5-5</a>
0x0_0004	Reserved	—	—	—
0x0_0008	Alternate configuration base address register (ALTCBAR)	R/W	0x0000_0000	<a href="#">5.1.4.2/5-7</a>
0x0_000C–0x0_001C	Reserved	—	—	—
0x0_0020	eLBC local access window 0 base address register (LBLAWBAR0)	R/W	0x0000_0000 <sup>1</sup>	<a href="#">5.1.4.3/5-7</a>
0x0_0024	eLBC local access window 0 attribute register (LBLAWAR0)	R/W	0x0000_0000 <sup>2</sup>	<a href="#">5.1.4.4/5-8</a>
0x0_0028	eLBC local access window 1 base address register (LBLAWBAR1)	R/W	0x0000_0000	<a href="#">5.1.4.3/5-7</a>
0x0_002C	eLBC local access window 1 attribute register (LBLAWAR1)	R/W	0x0000_0000	<a href="#">5.1.4.4/5-8</a>

Table 5-4. Local Access Register Memory Map (continued)

Local Access—Block Base Address 0x0_0000				
Local Memory Offset (Hex)	Register	Access	Reset	Section/Page
0x0_0030	eLBC local access window 2 base address register (LBLAWBAR2)	R/W	0x0000_0000	5.1.4.3/5-7
0x0_0034	eLBC local access window 2 attribute register (LBLAWAR2)	R/W	0x0000_0000	5.1.4.4/5-8
0x0_0038	eLBC local access window 3 base address register (LBLAWBAR3)	R/W	0x0000_0000	5.1.4.3/5-7
0x0_003C	eLBC local access window 3 attribute register (LBLAWAR3)	R/W	0x0000_0000	5.1.4.4/5-8
0x0_0040–0x0_0063	Reserved	—	—	—
0x0_0064	Reserved	—	—	—
0x0_0068–0x0_007C	Reserved	—	—	—
0x0_0080	PCI Express local access window base address register (PCIEXP1LAWBAR)	R/W	0x0000_0000	5.1.4.5/5-9
0x0_0084	PCI Express local access window attribute register (PCIEXP1LAWAR)	R/W	0x0000_0000	5.1.4.6/5-10
0x0_0088–0x0_009C	Reserved	—	—	—
0x0_00A0	DDR2 local access window 0 base address register (DDRLAWBAR0)	R/W	0x0000_0000 <sup>3</sup>	5.1.4.7/5-11
0x0_00A4	DDR2 local access window 0 attribute register (DDRLAWAR0)	R/W	0x0000_0000 <sup>4</sup>	5.1.4.8/5-12
0x0_00A8	DDR2 local access window 1 base address register (DDRLAWBAR1)	R/W	0x0000_0000	5.1.4.7/5-11
0x0_00AC	DDR2 local access window 1 attribute register (DDRLAWAR1)	R/W	0x0000_0000	5.1.4.8/5-12
0x0_00B0–0x0_00FC	Reserved	—	—	—

<sup>1</sup> Depends on reset configuration word high values. See [Section 5.1.4.3.1](#), “LBLAWBAR0[BASE\_ADDR] Reset Value,” for details.

<sup>2</sup> Depends on reset configuration word high values. See [Section 5.1.4.4.1](#), “LBLAWAR0[EN] and LBLAWAR0[SIZE] Reset Value,” for details.

<sup>3</sup> Depends on reset configuration word high values. See [Section 5.1.4.7.1](#), “DDRLAWBAR0[BASE\_ADDR] Reset Value,” for details.

<sup>4</sup> Depends on reset configuration word high values. See [Section 5.1.4.8.1](#), “DDRLAWAR0[EN] and DDRLAWAR0[SIZE] Reset Value,” for details.

## 5.1.4 Local Access Register Descriptions

This section describes the local access registers.

### 5.1.4.1 Internal Memory Map Registers Base Address Register (IMMRBAR)

The IMMR window contains configuration, control, and status registers, as well as internal device memory arrays. The internal memory map occupies a 1-Mbyte region of memory space. Its location is programmable using the internal memory map register (IMMR). The default base address for the internal

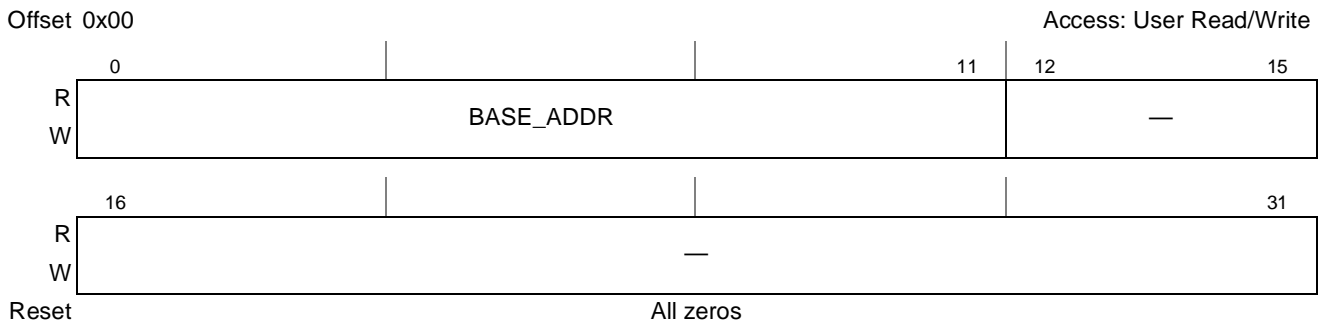
memory map register is 0xFF40\_0000. Because IMMRBAR is at offset 0x0 from the beginning of the local access registers, the IMMRBAR always points to itself.

#### 5.1.4.1.1 Updating IMMRBAR

Updates to IMMRBAR that relocate the entire 1-Mbyte region of the internal memory block 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, the following guidelines should be followed:

- IMMRBAR should be updated during initial configuration of the device when only one host or controller has access to the device as follows:
  - If the core is initializing the device, it should set IMMRBAR to the desired final location before enabling other I/O devices to access the device.
- When the e300 core is writing to IMMRBAR, it should use the following sequence:
  - Read the current value of IMMRBAR using a load word instruction followed by an **isync**. This forces all accesses to configuration space to complete.
  - Write the new value to IMMRBAR.
  - 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 IMMRBAR from its new location, followed by another **isync**.

The IMMRBAR is shown in [Figure 5-2](#).



**Figure 5-2. Internal Memory Map Registers' Base Address Register (IMMRBAR)**

[Table 5-5](#) defines the bit fields of IMMRBAR.

**Table 5-5. IMMRBAR Bit Settings**

Bits	Name	Description
0–11	BASE_ADDR	Identifies the 12 most-significant address bits of the base of the 1-Mbyte internal memory window.
12–31	—	Reserved. Software must write all zeros.

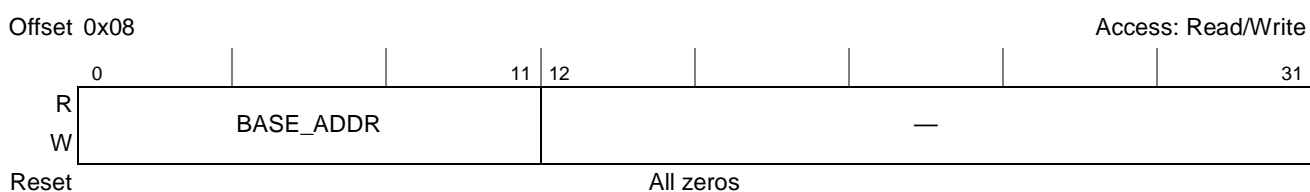
### 5.1.4.2 Alternate Configuration Base Address Register (ALTCBAR)

The alternate configuration base address register (ALTCBAR) is used to define the base address for an alternate 1-Mbyte region of configuration space to be used by the boot sequencer. 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 32-bit address. Thus, by configuring this register, the boot sequencer has access to the entire memory map, one 1-Mbyte block at a time. See [Section 17.4.5, “Boot Sequencer Mode,”](#) for more information.

#### NOTE

ALTCBAR is not considered a local access window on its own, so the boot sequencer must configure one of the other eight local access windows properly to reach the desired target peripherals.

The alternate configuration base address register is shown in [Figure 5-3](#).



**Figure 5-3. Alternate Configuration Base Address Register (ALTCBAR)**

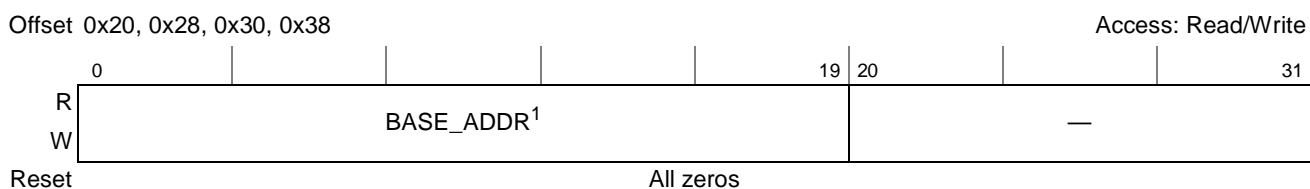
[Table 5-6](#) defines the bit fields of ALTCBAR.

**Table 5-6. ALTCBAR Bit Settings**

Bits	Name	Description
0–11	BASE_ADDR	Identifies the 12 most-significant address bits of an alternate base address used for boot sequencer configuration accesses.
12–31	—	Reserved. Write has no effect, read returns 0.

### 5.1.4.3 LBC Local Access Window *n* Base Address Registers (LBLAWBAR0–LBLAWBAR3)

The LBC local access window *n* base address registers (LBLAWBAR0–LBLAWBAR3) are shown in [Figure 5-4](#).



**Figure 5-4. LBC Local Access Window *n* Base Address Registers (LBLAWBAR0–LBLAWBAR3)**

<sup>1</sup> The LBLAWBAR0[BASE\_ADDR] reset value depends on the reset configuration word high values. See [Section 5.1.4.3.1, “LBLAWBAR0\[BASE\\_ADDR\] Reset Value,”](#) for a detailed description.

Table 5-7 defines the bit fields of LBLAWBAR0–LBLAWBAR3.

Table 5-7. LBLAWBAR0–LBLAWBAR3 Bit Settings

Bits	Name	Description
0–19	BASE_ADDR	Identifies the 20 most-significant address bits of the base of local access window <i>n</i> . The specified base address should be aligned to the window size, as defined by LBLAWARn[SIZE].
20–31	—	Reserved. Write has no effect, read returns 0.

#### 5.1.4.3.1 LBLAWBAR0[BASE\_ADDR] Reset Value

The core may also use a local bus peripheral device to fetch its boot vector. For this purpose, the LBLAWBAR0[BASE\_ADDR] reset value is set according to the value set in the reset configuration word high BMS field.

Table 5-8 defines the reset value of LBLAWBAR0[BASE\_ADDR].

Table 5-8. LBLAWBAR0[BASE\_ADDR] Reset Value

RCWHR[BMS]	BASE_ADDR Reset Value
0	0x00000
1	0xFF800

#### 5.1.4.4 LBC Local Access Window *n* Attributes Registers (LBLAWAR0–LBLAWAR3)

The LBC local access window *n* attributes registers (LBLAWAR0–LBLAWAR3) are shown in Figure 5-5.

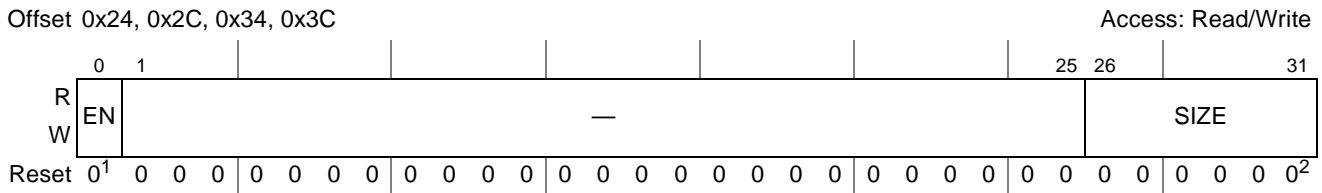


Figure 5-5. LBC Local Access Window *n* Attributes Registers (LBLAWAR0–LBLAWAR3)

- <sup>1</sup> The LBLAWAR0[EN] reset value depends on the reset configuration word high values. See Section 5.1.4.4.1, “LBLAWAR0[EN] and LBLAWAR0[SIZE] Reset Value,” for a detailed description.
- <sup>2</sup> The LBLAWAR0[SIZE] reset value is always 0b010110, meaning an 8-Mbyte local access window. See Section 5.1.4.4.1, “LBLAWAR0[EN] and LBLAWAR0[SIZE] Reset Value,” for a detailed description.

Table 5-9 defines the bit fields of LBLAWAR0–LBLAWAR3.

Table 5-9. LBLAWAR0–LBLAWAR3 Bit Settings

Bits	Name	Description
0	EN	0 Local bus local access window <i>n</i> is disabled. 1 Local bus local access window <i>n</i> is enabled and other LBLAWAR0 and LBLAWBAR0 fields combine to identify an address range for this window.

**Table 5-9. LBLAWAR0–LBLAWAR3 Bit Settings (continued)**

1–25	—	Reserved. Write has no effect, read returns 0.
26–31	SIZE	Identifies the size of the window from the starting address. Window size is $2^{(SIZE+1)}$ bytes. 000000–001010 Reserved. Window is undefined. 001011 4 Kbytes 001100 8 Kbytes 001101 16 Kbytes ..... $2^{(SIZE+1)}$ bytes 011110 2 Gbytes 011111–111111 Reserved. Window is undefined.

#### 5.1.4.4.1 LBLAWAR0[EN] and LBLAWAR0[SIZE] Reset Value

The core may use a local bus peripheral device to fetch its boot vector. For this purpose an 8-Mbyte ( $2^{(22+1)}$ ) local access window is defined by the LBLAWBAR0[SIZE] reset value, and LBLAWAR0 is enabled according to the value set in the reset configuration word high ROMLOC field.

Table 5-10 defines the reset value for LBLAWAR0[EN].

**Table 5-10. LBLAWAR0[EN] Reset Value**

RCWHR[ROMLOC]	RLEXT <sup>1</sup>	LBLAWAR0[EN] Reset Value	Description
000	01	1	e300 core boot not performed from a local bus device.
101	00		
	01		
110	00	0	e300 core boot performed from a local bus device. Local bus 8-Mbyte ( $2^{(22+1)}$ ) local access window is enabled.
others			

<sup>1</sup> For more information, see Section 4.3.2.2, “Reset Configuration Word High Register (RCWHR).”

#### 5.1.4.5 PCI Express Local Access Window Base Address Register (PCIEXP1LAWBAR)

The PCI Express 1 local access window base address register is shown in Figure 5-6.

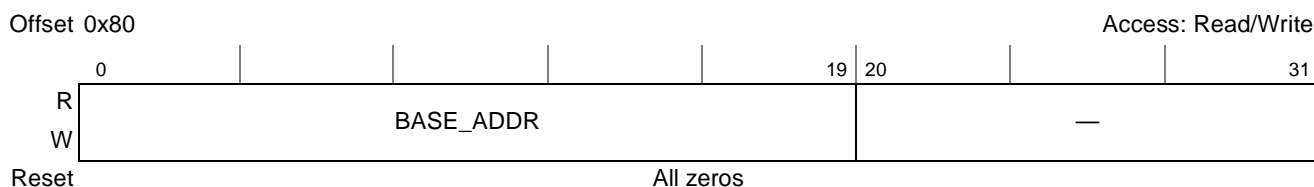
**Figure 5-6. PCI Express 1 Local Access Window Base Address Register (PCIEXP1LAWBAR)**

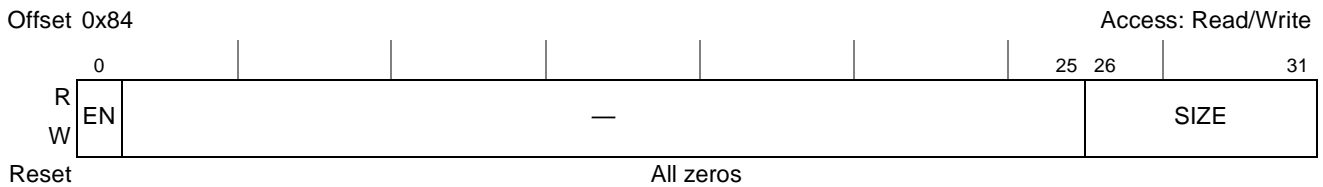
Table 5-11 defines the bit fields of PCIEXP1LAWBAR.

**Table 5-11. PCIEXP1LAWBAR Bit Settings**

Bits	Name	Description
0–19	BASE_ADDR	Identifies the 20 most-significant address bits of the base of local access window. The specified base address should be aligned to the window size, as defined by PCIEXP1LAWAR[SIZE].
20–31	—	Reserved. Write has no effect, read returns 0.

### 5.1.4.6 PCI Express Local Access Window Attributes Registers (PCIEXP1LAWAR)

The PCI Express 1 local access window attributes register is shown in Figure 5-7.



**Figure 5-7. PCI Express 1 Local Access Window Attributes Register (PCIEXP1LAWAR)**

Table 5-12 defines the bit fields of PCIEXP1LAWAR.

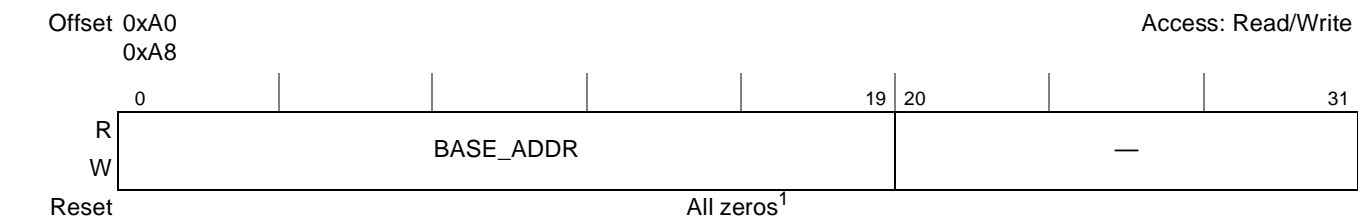
**Table 5-12. PCIEXP1LAWAR Bit Settings**

Bits	Name	Description
0	EN	0 The PCI Express 1 local access window is disabled. 1 The PCI Express 1 local access window is enabled and other PCIEXP1LAWAR fields combine to identify an address range for this window.
1–25	—	Reserved. Write has no effect, read returns 0.
26–31	SIZE	Identifies the size of the window from the starting address. Window size is $2^{(SIZE+1)}$ bytes. 000000–001010 Reserved. Window is undefined. 001011 4 Kbytes 001100 8 Kbytes 001101 16 Kbytes ..... $2^{(SIZE+1)}$ bytes 011110 2 Gbytes 011111–111111 Reserved. Window is undefined.



### 5.1.4.7 DDR Local Access Window $n$ Base Address Registers (DDRLAWBAR0–DDRLAWBAR1)

The DDR local access window  $n$  base address registers (DDRLAWBAR0–DDRLAWBAR1) are shown in [Figure 5-8](#).



<sup>1</sup> The reset value of DDRLAWBAR0[BASE\_ADDR] depends on the reset configuration word high values. See [Section 5.1.4.7.1, “DDRLAWBAR0\[BASE\\_ADDR\] Reset Value,”](#) for a detailed description

**Figure 5-8. DDR Local Access Window  $n$  Base Address Registers (DDRLAWBAR0–DDRLAWBAR1)**

[Table 5-13](#) defines the bit fields of DDRLAWBAR0–DDRLAWBAR1.

**Table 5-13. DDRLAWBAR0–DDRLAWBAR1 Bit Settings**

Bits	Name	Description
0–19	BASE_ADDR	Identifies the 20 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 DDRLAWAR $n$ [SIZE].
20–31	—	Reserved. Write has no effect, read returns 0.

#### 5.1.4.7.1 DDRLAWBAR0[BASE\_ADDR] Reset Value

The core may use a DDR SDRAM device to fetch its boot vector. For this purpose, the DDRLAWBAR0[BASE\_ADDR] reset value is set according to the value set in the reset configuration word high BMS field.

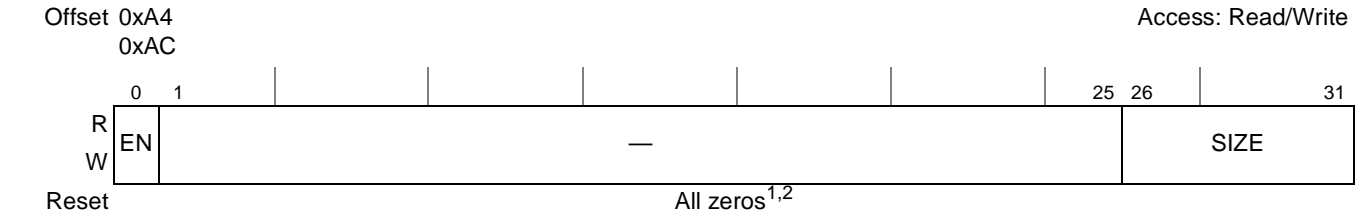
[Table 5-14](#) defines the reset value DDRLAWBAR0.

**Table 5-14. DDRLAWBAR0[BASE\_ADDR] Reset Value**

RCWHR[BMS]	DDRLAWBAR0[BASE_ADDR] Reset Value
0	0x00000
1	0xFF800

### 5.1.4.8 DDR Local Access Window $n$ Attributes Registers (DDRLAWAR0–DDRLAWAR1)

The DDR local access window  $n$  attributes registers (DDRLAWAR0–DDRLAWAR1) are shown in Figure 5-9.



- <sup>1</sup> The reset value of DDRLAWAR0[EN] depends on the reset configuration word high values. See Section 5.1.4.8.1, “DDRLAWAR0[EN] and DDRLAWAR0[SIZE] Reset Value,” for a detailed description.
- <sup>2</sup> The reset value of DDRLAWAR0[SIZE] is always 0b010110, meaning an 8-Mbyte local access window. See Section 5.1.4.8.1, “DDRLAWAR0[EN] and DDRLAWAR0[SIZE] Reset Value,” for a detailed description.

**Figure 5-9. DDR Local Access Window  $n$  Attributes Registers (DDRLAWAR0–DDRLAWAR1)**

Table 5-15 defines the bit fields of DDRLAWAR0–DDRLAWAR1.

**Table 5-15. DDRLAWAR0–DDRLAWAR1 Bit Settings**

Bits	Name	Description
0	EN	0 The DDR local access window $n$ is disabled. 1 The DDR local access window $n$ is enabled and other DDRLAWAR $n$ and DDRLAWBAR $n$ fields combine to identify an address range for this window.
1–25	—	Reserved. Write has no effect, read returns 0.
26–31	SIZE	Identifies the size of the window from the starting address. Window size is $2^{(SIZE+1)}$ bytes. 000000–001010 Reserved. Window is undefined. 001011 4 Kbytes 001100 8 Kbytes 001101 16 Kbytes ..... $2^{(SIZE+1)}$ bytes 011110 2 Gbytes 011111–111111 Reserved. Window is undefined.

#### 5.1.4.8.1 DDRLAWAR0[EN] and DDRLAWAR0[SIZE] Reset Value

The core may use a DDR SDRAM device to fetch its boot vector. For this purpose an 8-Mbyte ( $2^{(22+1)}$ ) local access window is defined by DDRLAWBAR0[SIZE] reset value, and DDRLAWAR0 is enabled according to the value set in the reset configuration word high ROMLOC field.

Table 5-16 defines the reset value DDRLAWAR0[EN] and DDRLAWAR0[SIZE].

**Table 5-16. DDRLAWAR0[EN] Reset Value**

RCWHR[ROMLOC]	RLEXT <sup>1</sup>	DDRLAWAR0[EN] Reset Value	Description
000	00	1	e300 core boot performed from a DDR SDRAM device. DDR 8-Mbyte ( $2^{(22+1)}$ ) local access window is enabled.
000	01	0	e300 core boot not performed from a DDR SDRAM device
Else	–	0	e300 core boot not performed from a DDR SDRAM device.

<sup>1</sup> For more information, see Section 4.3.2.2, “Reset Configuration Word High Register (RCWHR).”

## 5.1.5 Precedence of Local Access Windows

If two local access windows overlap, the lower numbered window takes precedence (see Table 5-1 for window numbers). For instance, if two windows are set up as shown in Table 5-17, local access window 1 governs the mapping of the 1-Mbyte region from 0x7FF0\_0000 to 0x7FFF\_FFF, even though the window described in local access window 7 also encompasses that memory region.

**Table 5-17. Overlapping Local Access Windows**

Window	Base Address	Size	Target Interface
1	0x7FF0_0000	1 Mbyte	Local bus
7	0x0000_0000	2 Gbytes	DDR SDRAM

## 5.1.6 Configuring Local Access Windows

After a local access window is enabled, it should not be modified while any device in the system may be using the window. Accordingly, a new window should not 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 bus local access windows 1–3 are being configured in order during the initialization process, the last write (to LBLAWAR3) should be followed by a read of LBLAWAR3 before any devices try to use any of these windows. If the configuration is being done by the local e300 core, the read of LBLAWAR3 should be followed by an **isync** instruction.

## 5.1.7 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 device internal interconnects from the transaction’s source to its target. Once 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. The local bus controller has base registers that perform a similar function.

The PCI Express interface has outbound address translation 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 windows. A single local access window can be further decoded to any number of chip-selects or to any number or outbound windows at the target interface.

## 5.1.8 Outbound Address Translation and Mapping Windows

Outbound address translation and mapping refers to the translation of addresses from the local 32-bit address space to the external address space and attributes of a particular I/O interface. On this device, the PCI Express block has an outbound address translation unit.

## 5.1.9 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 Express address space) to the local address space understood by the internal interfaces of this processor. It also refers to the mapping of transactions to a particular target interface and the assignment of transaction attributes. The PCI Express controller has inbound address translation unit.

### 5.1.9.1 PCI Express Inbound Windows

The PCI Express controller has four inbound windows. In the PCI Express EP (End Point) mode, these windows are same as the base address registers in the PCI Express programming model. In the PCI Express RC (Root Complex) mode, there are four separate registers, PEX\_RCIWBARL[0:3]. For more information, see [Section 14.6.1.1, “Address Translation Windows \(ATMUs\).”](#)

## 5.1.10 Internal Memory Map

All of the memory-mapped configuration, control, and status registers in the device are contained within a 1-Mbyte address region, referred as the IMMR. To allow for flexibility, the internal memory map block can be relocated in the local address space. The local address map location of this register block is controlled by the internal memory map registers' base address register (IMMRBAR); see [Section 5.1.4.1, “Internal Memory Map Registers Base Address Register \(IMMRBAR\).”](#) The default value for the IMMRBAR is 0xFF40\_0000.

### NOTE

The internal memory map window is always the highest priority local access window.

## 5.1.11 Accessing Internal Memory from External Masters

In addition to being accessible by the e300 processor, the IMMR memory window is accessible from external interfaces. This allows external masters on the I/O ports to configure the device.

External masters do not need to know the location of the IMMR 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's programming model that is accessible to the external master from its external memory map.

## 5.2 System Configuration

The following sections describe some general information and configuration options that affect system behavior and performance.

### 5.2.1 System Configuration Register Memory Map

Table 5-18 shows the memory map for the system configuration registers.

**Table 5-18. System Configuration Register Memory Map**

Local Memory Offset (Hex)	Register	Access	Reset	Section/Page
<b>System Configuration—Block Base Address 0x0_0000</b>				
0x00100	System general purpose register low (SGPRL)	R/W	0x0000_0000	5.2.2.1/5-16
0x00104	System general purpose register high (SGPRH)	R/W	0x0000_0000	5.2.2.2/5-16
0x00108	System part and revision ID register (SPRIDR)	R	0x8101_0010	5.2.2.3/5-17
0x0010C	Reserved	—	—	—
0x00110	System priority configuration register (SPCR)	R/W	0x0000_0000	5.2.2.4/5-17
0x00114	System I/O configuration register low (SICRL)	R/W	0x0000_0000 <sup>1</sup>	5.2.2.5/5-19
0x00118	System I/O configuration register high (SICRH)	R/W	0x0014_5000 <sup>2</sup>	5.2.2.6/5-22
0x0011C–0x00124	Reserved	—	—	—
0x00128	DDR control driver register (DDRCDR)	R/W	0x7304_0001	5.2.2.9/5-26
0x0012C	DDR debug status register (DDRDSR)	R	0x3300_0000	5.2.2.10/5-27
0x00140	PCI Express control register 1 (PECR1)	R/W	0x0000_0000	5.2.2.11/55-28
0x00144	eSDHC control register (SDHCCR)	R/W	0x0000_0000	5.2.2.12/55-29
0x00148	RTC control register (RTCCR)	R/W	0x0000_0000	5.2.2.13/55-31
0x00160–0x001FC	Reserved	—	—	—

<sup>1</sup> Bit #25 depends on the RCW.

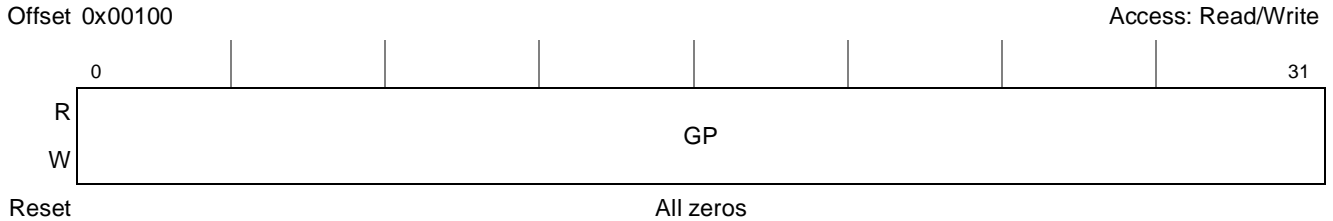
<sup>2</sup> Bit #30 depend on RCW.

## 5.2.2 System Configuration Registers

This section discusses the system configuration registers.

### 5.2.2.1 System General Purpose Register Low (SGPRL)

The system general purpose register low (SGPRL), shown in [Figure 5-10](#), can be used by software for any purpose. The values set in this register have no effect on the internal hardware.



**Figure 5-10. System General Purpose Register Low (SGPRL)**

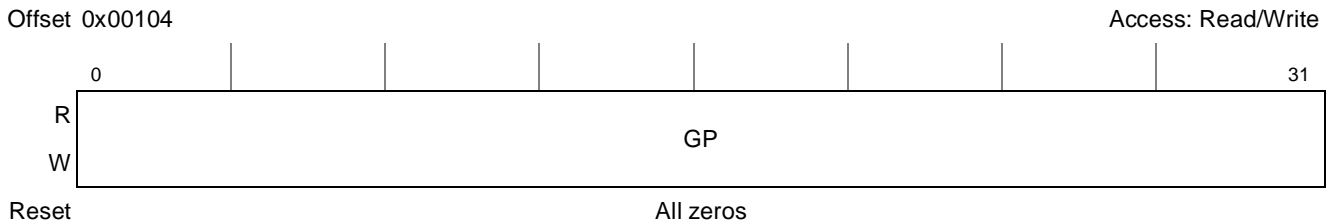
[Table 5-19](#) defines the bit fields of SGPRL.

**Table 5-19. SGPRL Bit Settings**

Bits	Name	Description
0–31	GP	General purpose

### 5.2.2.2 System General Purpose Register High (SGPRH)

The system general purpose register high (SGPRH), shown in [Figure 5-11](#), can be used by software for any purpose. The values set in this register have no effect on the internal hardware.



**Figure 5-11. System General Purpose Register High (SGPRH)**

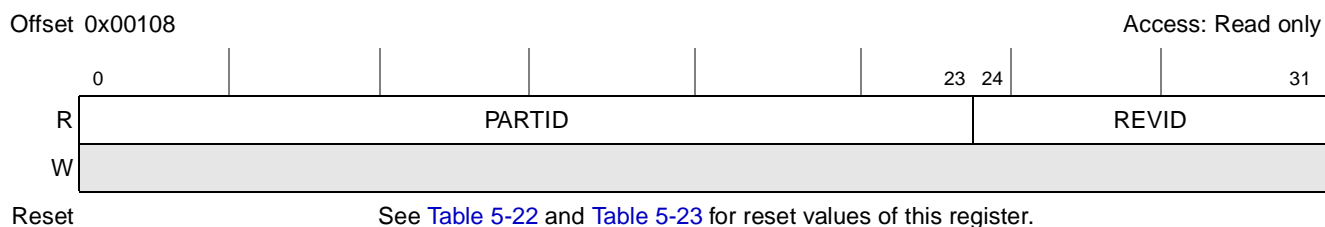
[Table 5-20](#) defines the bit fields of SGPRH.

**Table 5-20. SGPRH Bit Settings**

Bits	Name	Description
0–31	GP	General purpose

### 5.2.2.3 System Part and Revision ID Register (SPRIDR)

The SPRIDR, shown in [Figure 5-12](#), provides information about the device and revision numbers.



**Figure 5-12. System Part and Revision ID Register (SPRIDR)**

[Table 5-21](#) defines the bit fields of SPRIDR.

**Table 5-21. SPRIDR Bit Settings**

Bits	Name	Description
0–23	PARTID	Part identification. This read-only field is mask-programmed with a code corresponding to the device number. It is intended to help factory test and user code that is sensitive to device changes. The device number changes according to manufacturing considerations. See <a href="#">Table 5-22</a> for values of this field.
24–31	REVID	Revision identification. This read-only field is mask-programmed with a code corresponding to the revision number of the part defined in PARTID field. It is intended to help factory test and user code that is sensitive to device changes. The mask number is programmed in a commonly changed layer, and changes with each mask set change. See <a href="#">Table 5-23</a> for values of this field.

#### 5.2.2.3.1 SPRIDR[PARTID] Coding

[Table 5-22](#) defines the reset values of SPRIDR[PARTID].

**Table 5-22. PARTID Coding**

PARTID	Device Name	Package Type
0x810101	MPC8308	MAPBGA

#### 5.2.2.3.2 SPRIDR[REVID] Coding

[Table 5-23](#) defines the reset values of SPRIDR[REVID].

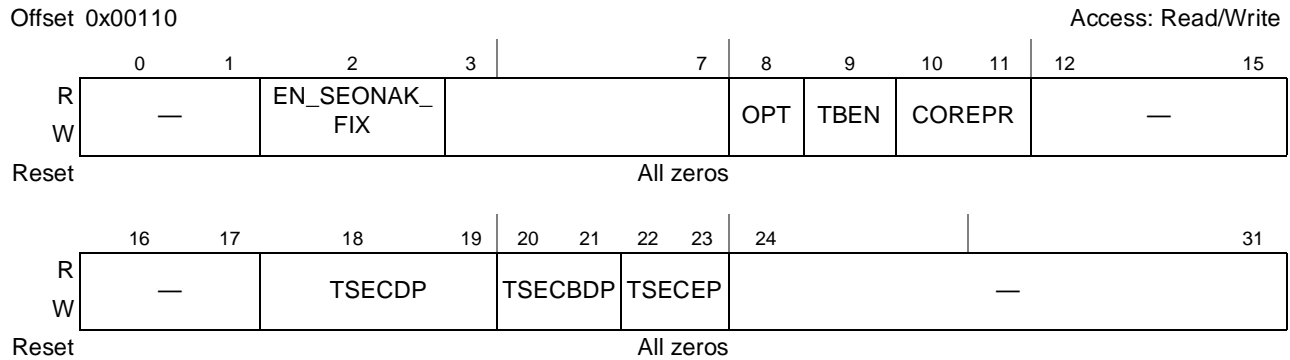
**Table 5-23. REVID Coding**

REVID	Device Revision
0x10	1.0

### 5.2.2.4 System Priority and Configuration Register (SPCR)

The system priority and configuration register (SPCR), shown in [Figure 5-13](#), controls the priority of requests for transactions on the internal system bus. This priority is considered by the system arbiter

whenever an internal unit requests mastership of the coherent system bus (CSB). The SPCR also includes some other control functions.



**Figure 5-13. System Priority Configuration Register (SPCR)**

Table 5-24 defines the bit fields of SPCR.

**Table 5-24. SPCR Bit Settings**

Bits	Name	Description
0–1	—	Reserved. Should be cleared.
2	EN_SEONAK_FIX	enable_se0nak_fix for USBDR If set, disables the SOF reporting when in SE0_NAK test mode.
3–7	—	Reserved. Should be cleared.
8	OPT	Optimize. Setting this bit may enhance the performance of transactions issued to the internal coherent system bus (CSB) by a master such as USB controller. Performance is enhanced by reading more bytes on the bus than actually needed by the master in the case that this is more efficient. The user may set this bit only if it is known that USB transactions sent to the internal CSB are not accessing devices in which speculative reads may change the state of the device (for example, FIFOs in which reading a byte may advance some internal counter). 0 No performance enhancement. 1 Performance enhancement by speculative reading is enabled.
9	TBEN	e300 core time base unit enable 0 Time base unit is disabled. 1 Time base unit is enabled.
10–11	COREPR	e300 core CSB request priority. The priority level for the core in accessing the CSB can be chosen from four possible levels. 00 Level 0 (lowest priority) 01 Level 1 10 Level 2 11 Level 3 (highest priority)
12–17	—	Reserved. Should be cleared
18–19	TSECDP	eTSEC Data Priority. Selects the CSB request priority driven by eTSEC1 and eTSEC2 when it requires to transfer data on this bus. The level of priority can be chosen from four possible levels. 00 Level 0 (lowest priority) 01 Level 1 10 Level 2 11 Level 3 (highest priority)



Table 5-24. SPCR Bit Settings (continued)

Bits	Name	Description
20–21	TSECBDP	eTSEC buffer descriptor priority. Selects the CSB request priority driven by eTSEC1 and eTSEC2 when they require to transfer a buffer descriptor (BD) on this bus. The level of priority can be chosen from four possible levels. 00 Level 0 (lowest priority) 01 Level 1 10 Level 2 11 Level 3 (highest priority)
22–23	TSECEP	TSEC emergency priority. Selects the CSB request priority driven by eTSEC1 and eTSEC2 when an emergency condition occurs. The level of priority can be chosen from four possible levels. 00 Level 0 (lowest priority) 01 Level 1 10 Level 2 11 Level 3 (highest priority)
24–31	—	Reserved. Should be cleared.

### 5.2.2.5 System I/O Configuration Register Low (SICRL)

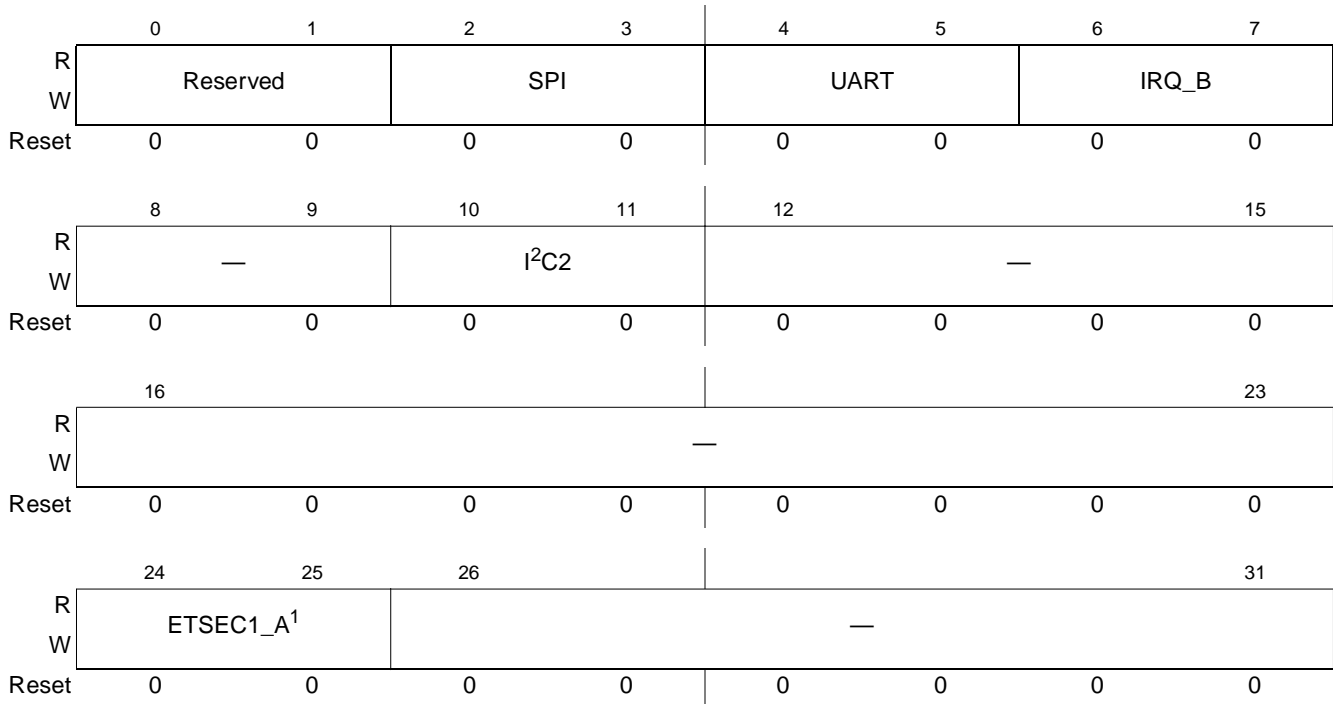
The system I/O configuration register low (SICRL) controls the multiplexing of some of the device I/O pins. Each bit or set of bits in the SICRL selects which function is used by a certain group of the device pins.

The reset value of this register depends on TSEC1M fields setting in the reset configuration word high. The function of these pins can be changed by writing to this register during system initialization. TSOBI1 reset value also depends on the TSEC1M field settings in the reset configuration word high in order to select the correct output buffer impedance for full or reduced TSEC pin mode.

Figure 5-14 shows SICRL.

Offset 0x00114

Access: Read/Write



<sup>1</sup> Bit #25 depends on the TSEC1M in the RCW. If it is set to RGMII, this bit is set to 1 on reset; for other values of TSEC1M, this is zero.

Figure 5-14. System I/O Configuration Register Low (SICRL)

Table 5-24 defines the bit fields of SICRL. Each Pin Function column lists the name of the multi-function pin used in this option. Some groups have only two options (shown as Pin Function 0 and Pin Function 1) and therefore, only one control bit. In this case they can only have a value of 0b0 or 0b1. Other groups may have four options (shown as Pin Function 0, Pin Function 1, Pin Function 2, and Pin Function 3) and therefore, two control bits. In this case they can have a value of 0b00, 0b01, 0b10, or 0b11. Use the notations ‘0bN’ or ‘0bNN’ according to whether a group has one or two control bits, respectively.

Table 5-25. SICRL Bit Settings

SICRL[Bits] Value		0b00	0b01	0b10	0b11	Reset
Bits	Group	Pin Function 0	Pin Function 1	Pin Function 2	Pin Function 3	
0-1	—	—	—	—	—	—
2-3	SPI <sup>1</sup>	SPI_MOSI	MSRCID4	—	LSRCID4	00
		SPI_MISO	MDVAL	—	LDVAL	
		SPICLK	—	—	—	
		SPISEL	—	—	—	

Table 5-25. SICRL Bit Settings (continued)

SICRL[Bits] Value		0b00	0b01	0b10	0b11	Reset
Bits	Group	Pin Function 0	Pin Function 1	Pin Function 2	Pin Function 3	
4–5	UART	UART_SOUT1	MSRCID0 (DDRID)	—	LSRCID0	00
		UART_SIN1	MSRCID1 (DDRID)	—	LSRCID1	
		UART_SOUT2	MSRCID2 (DDRID)	—	LSRCID2	
		UART_SIN2	MSRCID3 (DDRID)	—	LSRCID3	
6–7	$\overline{\text{IRQ}}$	$\overline{\text{IRQ}}[0]$	$\overline{\text{MCP\_IN}}$	—	—	00
		$\overline{\text{IRQ}}[1]$	$\overline{\text{MCP\_OUT}}$	—	—	
		$\overline{\text{IRQ}}[2]$	$\overline{\text{CKSTOP\_OUT}}$	—	—	
		$\overline{\text{IRQ}}[3]$	$\overline{\text{CKSTOP\_IN}}$	—	—	
8–9	—	—	—	—	—	—
10–11	I <sup>2</sup> C2	IIC_SDA2	$\overline{\text{CKSTOP\_OUT}}$	—	—	00
		IIC_SCL2	$\overline{\text{CKSTOP\_IN}}$	—	—	
12–15	—	—	—	—	—	—
16–19	—	—	—	—	—	00
20–23	—	—	—	—	—	—
24–25	ETSEC1_A	TSEC1_TX_CLK	TSEC1_GTX_CLK125	—	—	—
26–31	—	—	—	—	—	00

<sup>1</sup> When SICRL[SPI] = 0b01, SPICLK and SPISEL are in High-Z state.

### NOTE

An empty column cannot be used for this register. A function should be selected so that the column is non-empty.

### 5.2.2.6 System I/O Configuration Register High (SICRH)

The system I/O configuration register high (SICRH), shown in [Figure 5-15](#), controls the multiplexing of the rest of the device I/O pins. Each bit or set of bits in this register selects which function is used by a certain group of the device pins.

Offset 0x00118

Access: Read/Write

	0	1	2	3	4	5	6	7
R	eSDHC_A		eSDHC_B		eSDHC_C		GPIO_A	
W								
Reset	0	0	0	0	0	0	0	0
	8	9	10	11	12	13	14	15
R	GPIO_B		IEEE1588_A		USB		GTM	
W								
Reset	0	0	0	1	0	1	0	0
	16	17	18	19	20	22	23	
R	IEEE1588_B		ETSEC2		—		GPIO_SEL	
W								
Reset	0	1	0	1	0	0	0	0
	24	26	27	28	29	30	31	
R	—		TMROBI	—		TSOBI1	TSOBI2	
W								
Reset	0	0	0	0	0	0	depends on RCW	0

**Figure 5-15. System I/O Configuration Register High (SICRH)**

[Table 5-26](#) defines the bit fields of SICRH. Each Pin Function column lists the name of the multi-function pin used in this option. Note that the groups have two control bits to program the pin functionality.

Note that bits 30 and 31 (TSOBI1 and TSOBI2), which control TSEC output buffer impedance, are described in [Table 5-27](#).

**Table 5-26. SICRH Bit Settings**

SICRH[Bits] Value		0b00	0b01	0b10	0b11	Reset Value
Bits	Group	Pin Function 0	Pin Function 1	Pin Function 2	Pin Function 3	
0–1	eSDHC_A <sup>1</sup>	SD_CLK	—	—	GPIO_16	00
		SD_CMD	—	—	GPIO_17	
		$\overline{\text{SD\_CD}}$	GTM1_TIN1	—	GPIO_18	
		SD_WP	$\overline{\text{GTM1\_TGATE1}}$	—	GPIO_19	
2–3	eSDHC_B	SD_DAT0	$\overline{\text{GTM1\_TOUT1}}$	—	GPIO_20	00
		SD_DAT1	$\overline{\text{GTM1\_TOUT2}}$	—	GPIO_21	

Table 5-26. SICRH Bit Settings (continued)

SICRH[Bits] Value		0b00	0b01	0b10	0b11	Reset Value
Bits	Group	Pin Function 0	Pin Function 1	Pin Function 2	Pin Function 3	
4–5	eSDHC_C	SD_DAT2	GTM1_TIN2	—	GPIO_22	00
		SD_DAT3	GTM1_TGATE2	—	GPIO_23	
6–7	GPIO_A	GPIO_0	TSEC2_COL	—	—	00
		GPIO_1	TSEC2_TX_ER	—	—	
		GPIO_2	TSEC2_GTX_CLK	—	—	
		GPIO_3	TSEC2_RX_CLK	—	—	
		GPIO_4	TSEC2_RX_DV	—	—	
		GPIO_5	TSEC2_RXD[3]	—	—	
		GPIO_6	TSEC2_RXD[2]	—	—	
		GPIO_7	TSEC2_RXD[1]	—	—	
		GPIO_8	TSEC2_RXD[0]	—	—	
		GPIO_9	TSEC2_RX_ER	—	—	
		GPIO_11	TSEC2_TXD[3]	—	—	
		GPIO_12	TSEC2_TXD[2]	—	—	
		GPIO_13	TSEC2_TXD[1]	—	—	
		GPIO_14	TSEC2_TXD[0]	—	—	
GPIO_15	TSEC2_TX_EN	—	—			
8–9	GPIO_B	GPIO_10	TSEC2_TX_CLK	TSEC2_GTX_CLK125	—	00
10–11	IEEE1588_A	—	TSEC_TMR_GCLK	—	—	01
		—	TSEC_TMR_PP[1]	—	—	
		—	TSEC_TMR_PP[2]	—	—	
		—	TSEC_TMR_PP[1]	—	—	
		—	TSEC_TMR_PP[2]	—	—	
		—	TSEC_TMR_PP[3]	—	GPIO_13	
		—	TSEC_TMR_ALARM[1]	—	—	
		—	TSEC_TMR_ALARM[2]	—	GPIO_14	
		—	TSEC_TMR_CLK	—	GPIO_8	
		—	TSEC_TMR_TRIG[1]	—	GPIO_11	
—	TSEC_TMR_TRIG[2]	—	GPIO_12			

Table 5-26. SICRH Bit Settings (continued)

SICRH[Bits] Value		0b00	0b01	0b10	0b11	Reset Value
Bits	Group	Pin Function 0	Pin Function 1	Pin Function 2	Pin Function 3	
12–13	USB	—	USBDR_PWR_FAULT	—	—	01
		—	USBDR_CLK	—	—	
		—	USBDR_DIR	—	—	
		—	USBDR_NXT	—	—	
		—	USBDR_TXDRXD[0]	—	—	
		—	USBDR_TXDRXD[1]	—	—	
		—	USBDR_TXDRXD[2]	—	—	
		—	USBDR_TXDRXD[3]	—	—	
		—	USBDR_TXDRXD[4]	—	—	
		—	USBDR_TXDRXD[5]	—	—	
		—	USBDR_TXDRXD[6]	—	—	
		—	USBDR_TXDRXD[7]	—	—	
		—	USBDR_PCTL[0]	—	—	
		—	USBDR_PCTL[1]	—	—	
—	USBDR_STP	—	—			
14–15	GTM	—	$\overline{\text{GTM1\_TGATE3}}$	—	—	01
		—	GTM1_TIN4	—	—	
		—	$\overline{\text{GTM1\_TGATE4}}$	—	GPIO_15	
		—	GTM1_TIN3	—	—	
		—	$\overline{\text{GTM1\_TOUT3}}$	—	GPIO_9	
		—	$\overline{\text{GTM1\_TOUT4}}$	—	GPIO_10	
		—	—	—	GPIO_7 <sup>2</sup>	
16–17	IEEE1588_B	—	TSEC2_TMR_RX_ESFD	—	GPIO_1	01
		—	TSEC2_TMR_TX_ESFD	—	GPIO_2	
		—	TSEC1_TMR_RX_ESFD	—	GPIO_3	
		—	TSEC1_TMR_TX_ESFD	—	GPIO_4	
		—	—	—	GPIO_5 <sup>2</sup>	
		—	—	—	GPIO_6 <sup>2</sup>	
18–19	ETSEC2	—	TSEC2_CRS	—	GPIO_0	01
20–22	—	Reserved	—	—	—	—
23	GPIO_SEL	See <a href="#">Table 5-27</a> for description and reset value.				

Table 5-26. SICRH Bit Settings (continued)

SICRH[Bits] Value		0b00	0b01	0b10	0b11	Reset Value
Bits	Group	Pin Function 0	Pin Function 1	Pin Function 2	Pin Function 3	
24–26	—	Reserved	—	—	—	—
27	TMROBI	See Table 5-27 for description and reset value.				
28–29	—	Reserved	—	—	—	—
30	TSOBI1	See Table 5-27 for description and reset value.				
31	TSOBI2					

<sup>1</sup> When SICRH[eSDHC\_A] = 0b01, SD\_CLK and SD\_CMD are in High-Z state.

<sup>2</sup> To enable the GPIO functionality, this bit must be programmed to 11.

Table 5-27. SICRH[27–31] Bit Settings

Bits	Name	Description	Reset Value
23	GPIO_SEL	GPIO[0:15] is provided at two places. This bit selects the position of the GPIO registers. 0 Primary GPIO[0:15] provided as function-0, selectable through GPIO_A and GPIO_B. 1 GPIO[0:15] provided in function-3, selectable through GTM, IEEE1588_A, IEEE1588_B, and ETSEC2.	0
27	TMROBI	IEEE1588 timer TSEC_TMR ports output buffer impedance. This bit controls the output buffer impedance of the TMR output/input signals used for reduced pin mode interfaces (RGMII). The output buffer impedance should be correlated to the voltage supplied to the TSEC1 I/O pins (LVDD1). For non-eTSEC mode of operation, this bit must be cleared. 0 Output buffer is set for 40 Ω, 3.3 V. 1 Output buffer is set for 40 Ω, 2.5 V.	0
30	TSOBI1	TSEC1 output buffer impedance. This bit controls the output buffer impedance of the TSEC1 output signals, used for reduced pin mode interfaces (RGMII). The output buffer impedance should be correlated to the voltage supplied to the TSEC1 I/O pins (LVDD1). For non-eTSEC mode of operation, this bit must be cleared. 0 Output buffer is set for 40 Ω, 3.3 V. 1 Output buffer is set for 40 Ω, 2.5 V.	0 Else 1 RGMII <sup>1</sup>
31	TSOBI2	TSEC2 output buffer impedance. This bit controls the output buffer impedance of TSEC2 output signals, used for reduced pin mode interfaces (RGMII). The output buffer impedance should be correlated to the voltage supplied to the TSEC2 I/O pins (LVDD2). 0 Output buffer is set for 40 Ω, 3.3 V. 1 Output buffer is set for 40 Ω, 2.5 V.	0 Else 1 RGMII <sup>2</sup>

<sup>1</sup> If RCWH[ETSEC1M] is RGMII, the reset value is 1; otherwise, it is 0

<sup>2</sup> If RCWH[ETSEC2M] is RGMII, the reset value is 1; otherwise, it is 0.

### NOTE

An empty column cannot be used for this register. A function should be selected so that the column is non-empty.

### 5.2.2.7 Selection of Pin Functions During Reset

Few functions muxed with the I/Os are needed only during the period when the device is in reset state. Those functionality are selected by default during the reset phase. Once the device comes out of reset, the pad switches to the function needed for normal operation mode. The table below provides the list.

Pin Name	Function Selection during Reset	Normal Operation Mode
TSEC1_TX_ER	LB_POR_CFG_BOOT_ECC	TSEC1_TX_ER
TSEC1_TXD3	CFG_RESET_SOURCE[0]	TSEC1_TXD3
TSEC1_TXD2	CFG_RESET_SOURCE[1]	TSEC1_TXD2
TSEC1_TXD1	CFG_RESET_SOURCE[2]	TSEC1_TXD1
TSEC1_TXD0	CFG_RESET_SOURCE[3]	TSEC1_TXD0
TSEC1_TX_EN	LBC_PM_REF_10	TSEC1_TX_EN

### 5.2.2.8 Debug Configuration

Debug information may be driven on the device pins. This information can identify the internal source of a transaction that reached the DDR SDRAM or local bus interfaces. The device can be configured to drive the MSRCID[0:4] and MDVAL or LSRCID[0:4] and LDVAL signals, respectively, on other device pins. The coding of the source ID debug information is the same as the coding of the MSTR\_ID field in the AEATR register of the arbiter (See [Section 6.2.6, “Arbiter Event Attributes Register \(AEATR\).”](#)).

### 5.2.2.9 DDR Control Driver Register (DDRCDR)

The DDR control driver register (DDRCDR) contains bits that allow control over the driver of the DDR SDRAM controller.

DDRCDR is shown in [Figure 5-16](#).

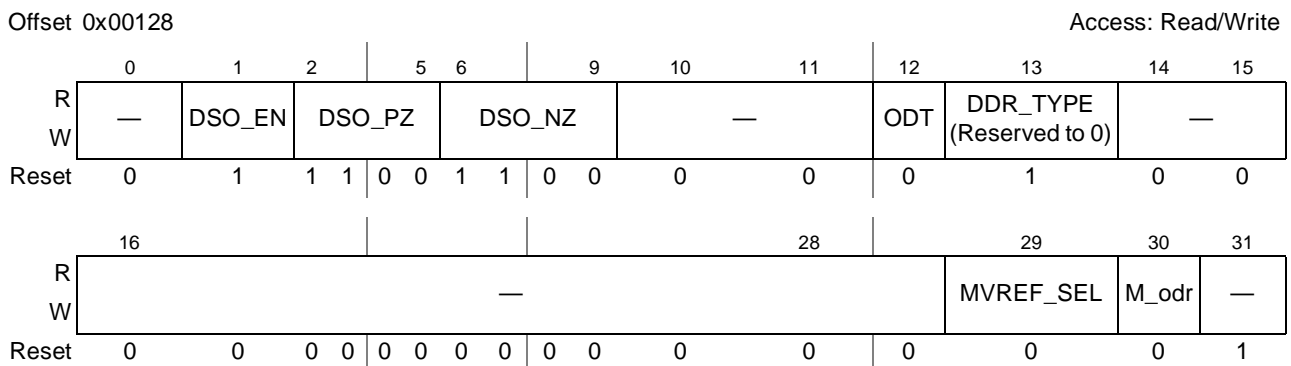


Figure 5-16. DDR Control Driver Register (DDRCDR)



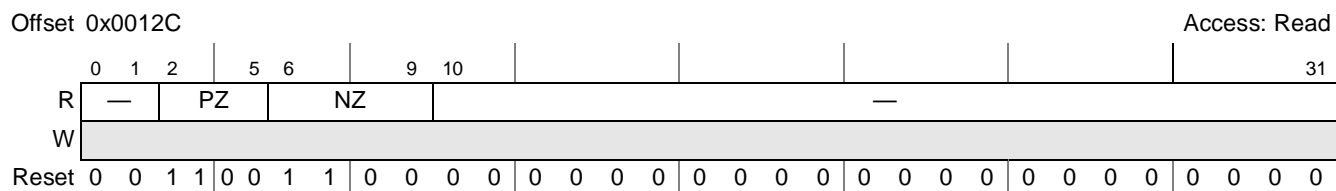
Table 5-28 shows the bit definition of the DDRCDR.

**Table 5-28. DDRCDR Field Descriptions**

Bits	Name	Description
0	—	Reserved
1	DSO_EN	0 DDR driver software override disable 1 DDR driver software override enable
2–5	DSO_PZ	DDR driver software p-impedance override 0000 Half strength—Highest Z 1000 Much higher Z than nominal 1100 Higher Z than nominal 1110 Nominal impedance setting 1111 Lower Z than nominal
6–9	DSO_NZ	DDR driver software n-impedance override 0000 Half strength—Highest Z 1000 Much higher Z than nominal 1100 Higher Z than nominal 1110 Nominal impedance setting 1111 Lower Z than nominal
10–11	—	Reserved. Should be cleared.
12	ODT	ODT termination value for I/Os 0 75 $\Omega$ 1 150 $\Omega$
13	DDR_TYPE	Selects voltage level for DDR pads 0 DDR2 (1.8V mode) nominal impedance—18 $\Omega$ 1 DDR1 (2.5V mode) nominal impedance—18 $\Omega$ <b>Note:</b> DDR_TYPE must be set according to the logical type of the DDR memory devices, as it affects logic behavior of the DDR controller as well as the physical parameters of the DDR I/O pads.
14–28	—	Reserved
29	MVREF_SEL	MVREF_SEL 0 MVREF is i/p from external source 1 MVREF is generated internally from GVDD
30	M_odr	Disable memory transaction reordering 0 Memory transaction reordering enabled 1 Memory transaction reordering disabled
31	—	Reserved

### 5.2.2.10 DDR Debug Status Register (DDRDSR)

Figure 5-17 contains the debug status bits from the DDR SDRAM controller.



**Figure 5-17. DDR Debug Status Register (DDRDSR)**

Table 5-29 shows the bit settings of the DDRDSR.

Table 5-29. DDRDSR Field Descriptions

Bits	Name	Description
0-1	—	Reserved
2-5	PZ	Current setting of PFET driver impedance 0000 Half strength—highest Z 1000 Higher Z than nominal 1100 Nominal impedance setting 1110 Lower Z than nominal 1111 Much lower Z than nominal
6-9	NZ	Current setting of NFET driver impedance 0000 Half strength—highest Z 1000 Higher Z than nominal 1100 Nominal impedance setting 1110 Lower Z than nominal 1111 Much lower Z than nominal
10-31	—	Reserved

### 5.2.2.11 PCI Express Control Registers (PECR1)

The PCI Express control registers can be used to control various settings that affect the system response to PCI Express controller DMA operations or PIO inbound operation. Those registers also control soft reset assertion and negation to various logic parts of the PCI Express controllers. Note that PECR1 programming controls the behavior of PCI Express controller for port 1. PECR1 is located at offset 0x140.

Figure 5-18 shows the PECR bit settings.

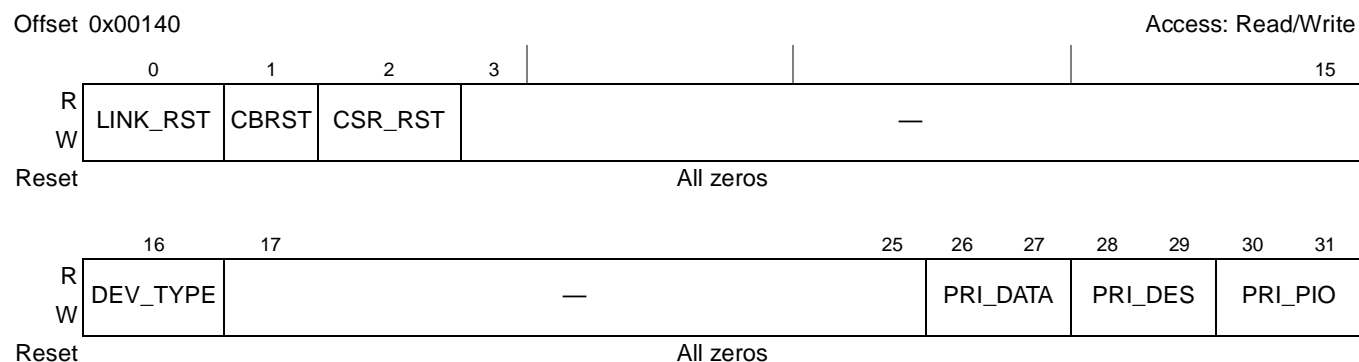


Figure 5-18. PCI Express Controller Registers (PECR1)

Downloaded from [Elcodis.com](http://Elcodis.com) electronic components distributor

Table 5-30 describes the bits of the PECR register.

**Table 5-30. PECR Field Description**

Bits	Name	Description
0	LINK_RST	Link soft reset (active low). Assert soft reset to PCI Express controller's logic that is related to the link, the configuration registers, and the core logic. Should be negated after device type is programmed and the SerDes initialization is completed. 0 Link soft reset is asserted 1 Link soft reset is negated
1	CBRST	CSB bridge soft reset (active low). Assert soft reset to PCI Express controller's logic related to the CSB bridge. Should be negated after device type is programmed and the SerDes initialization is completed. The CSB bridge soft reset should be asserted also when hot reset is detected, in order to clean the queues of the CSB interface. 0 CSB bridge soft reset is asserted 1 CSB bridge soft reset is negated
2	CSR_RST	CSR soft reset (active low). Assert soft reset to PCI Express controller's logic that is related to the CSR (control and status) registers. Should be negated after device type is programmed and the SerDes initialization is completed. 0 CSR soft reset is asserted 1 CSR soft reset is negated
3–15	—	Reserved
16	DEV_TYPE	Device type. Program device type. 0 EP (end point) 1 RC (root complex)
17–25	—	Reserved
26–27	PRI_DATA	Data priority. This field is used to present priority level for CSB arbitration for PCI Express controller's DMA requests. Bits 26–27 are used when the request belongs to data transfer. 00 Level 0 (lowest priority) 01 Level 1 10 Level 2 11 Level 3 (highest priority)
28–29	PRI_DES	DMA descriptor priority. This field is used to present priority level for CSB arbitration for PCI Express controller's DMA requests. Bits 28–29 are used when the request belongs to descriptor fetch or update. 00 Level 0 (lowest priority) 01 Level 1 10 Level 2 11 Level 3 (highest priority)
30–31	PRI_PIO	PIO priority. This field is used to present priority level for CSB arbitration for PCI Express controller's PIO inbound requests. 00 Level 0 (lowest priority) 01 Level 1 10 Level 2 11 Level 3 (highest priority)

### 5.2.2.12 eSDHC Control Registers (SDHCCR)

The eSDHC control registers can be used to control various settings that affect the priority and DMA operations. SDHCCR1 is located at offset 0x144.

Figure 5-19 shows the SDHCCR bit settings.

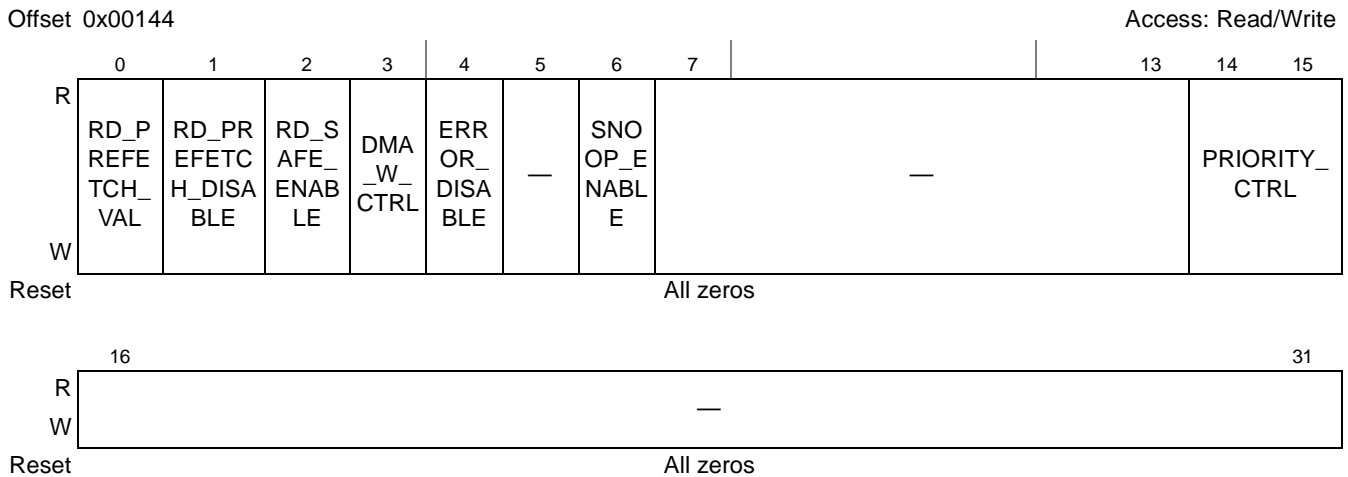


Figure 5-19. eSDHC Control Register (SDHCCR)

Table 5-31 describes the bits of the SDHCCR register.

Table 5-31. SDHCCR Field Description

Bits	Name	Description
0	RD_PREFETCH_VAL	This determines the prefetch byte count to be used if RD_PREFETCH_DISABLE is not set. 0 32 byte prefetch 1 64 byte prefetch
1	RD_PREFETCH_DISABLE	Read prefetch disable. This should be cleared 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. 0 It is allowed to prefetch data on DMA read operation 1 It is not allowed to prefetch data on DMA read operation
2	RD_SAFE_ENABLE	Read Safe enable. This bit should be set only 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 unaligned reading operation can be rounded up to enable more efficient read operations. 0 It is not safe to read more bytes that were intended 1 It is safe to read more bytes that were intended
3	DMA_W_CTRL	Write delay control bit. 0 No delay 1 Delay the Write/Read transaction till the interrupt is active
4	ERROR_DISABLE	Ignore or react to bus errors. 0 React to bus transaction errors 1 Ignore bus transaction errors
5	—	Reserved
6	SNOOP_ENABLE	Snoop attribute. 0 DMA transactions are not snooped by e300 CPU data cache 1 DMA transactions are snooped by e300 CPU data cache
7-13	Reserved	—

Table 5-31. SDHCCR Field Description (continued)

Bits	Name	Description
14–15	PRIORITY_CTRL	Priority. This field is used to present priority level for CSB arbitration for eSDHC DMA requests. 00 Level 0 (lowest priority) 01 Level 1 10 Level 2 11 Level 3 (highest priority)
16–31	Reserved	—

### 5.2.2.13 RTC Control Registers (RTCCR)

The RTC control register controls the reset for the RTC.

Figure 5-20 shows the RTCCR bit settings.

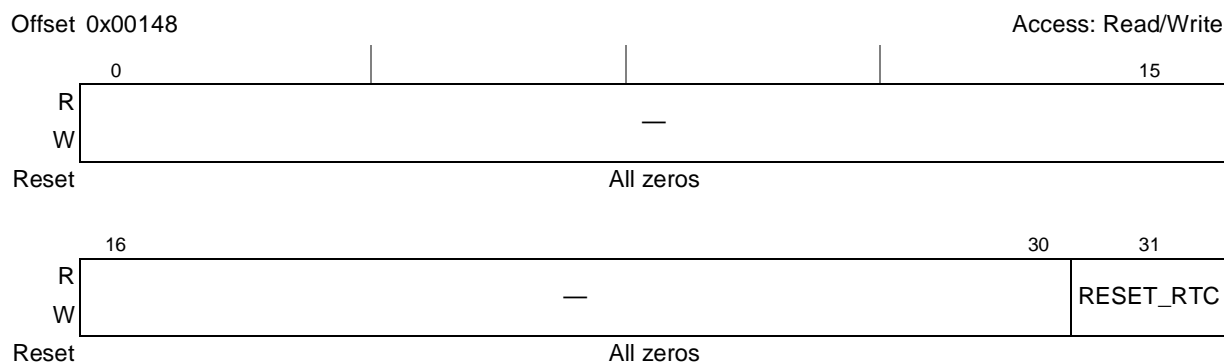


Figure 5-20. RTC Control Register (RTCCR)

Table 5-32 describes the bits of the RTCCR register.

Table 5-32. RTCCR Field Description

Bits	Name	Description
0–30	Reserved	—
31	RESET_RTC	Reset RTC 1 Software needs to assert this bit to reset the RTC logic and then negate this bit to allow the RTC logic to operate normally. Note that this register is reset by Power On Reset or HRESET, but RTC is not initialized during Power On Reset or HRESET. 0 Deassert reset to the RTC For more information, see <a href="#">Section 15.8.1.1, “RTC Reset Sequence.”</a>

## 5.3 Software Watchdog Timer (WDT)

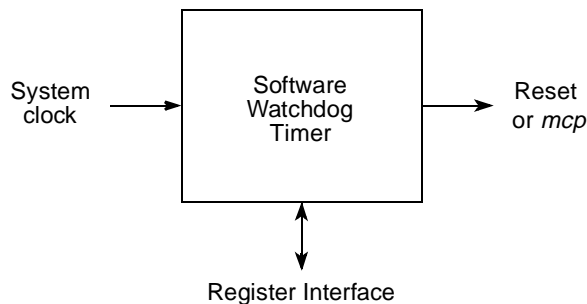
The following sections describe the theory of operation of the software watchdog timer (WDT) in the device, including a definition of the external signals and the functions they serve. Additionally, the configuration, control, and status registers are also described. Note that individual chapters in this book describe specific initialization aspects for each individual block.

### 5.3.1 WDT Overview

The device provides a software watchdog timer (WDT) feature to prevent system lock in case the software becomes trapped in loops with no controlled exit. Watchdog timer operations are configured in the system watchdog control register (SWCRR).

The watchdog counter is a free-running down-counter that generates a reset or a non-maskable interrupt on underflow. To prevent a reset, software must periodically restart the countdown. The WDT is responsible for asserting a hardware reset or machine-check interrupt (*mcp*) if the software fails to service the software watchdog timer for a certain period of time (for example, because software is lost or trapped in a loop with no controlled exit).

Figure 5-21 shows a high-level block diagram of the WDT.



**Figure 5-21. Software Watchdog Timer High-Level Block Diagram**

The software watchdog timer is enabled after reset to cause a hardware reset if it times out. The user has the option of disabling the software watchdog if it is not needed. If used, the software watchdog timer requires a special service sequence to be executed periodically. Without this periodic servicing, the software watchdog timer times out and issues a reset or a non-maskable interrupt.

### 5.3.2 WDT Features

The WDT includes the following key features:

- Based on 16-bit prescaler and 16-bit down-counter
- Provides a selectable range for the time-out period
- Provides ~34.36 sec maximum software time-out delay for 125-MHz input clock
- Functional and programming compatibility with MPC8260 watchdog timer

### 5.3.3 WDT Modes of Operation

The WDT unit can operate in the following modes:

- WDT enable/disable mode

If the software watchdog timer is not needed, the user can disable it with software after a system reset. When the watchdog timer is disabled, the watchdog counter and prescaler counter are held in a stopped state.

- WDT output reset/interrupt mode

Without software periodic servicing, the software watchdog timer times out and issues a reset or a nonmaskable interrupt (*mcp*)

- WDT prescaled/non-prescaled clock mode

The WDT counter clock can be prescaled by programming the SWCRR[SWPR] bit, which controls the divide-by-65,536 of the WDT counter.

### 5.3.4 WDT Memory Map/Register Definition

The WDT programmable register map occupies 16 bytes of memory-mapped space. Reading undefined portions of the memory map returns all zeros, and writing has no effect.

All WDT registers are 16- or 32-bits wide, located on 16-bit address boundaries, and should be accessed as 16- or 32-bit quantities. All addresses used in this chapter are offsets from the WDT base, as defined in Chapter 3, “Memory Map.”

Table 5-33 shows the WDT memory map.

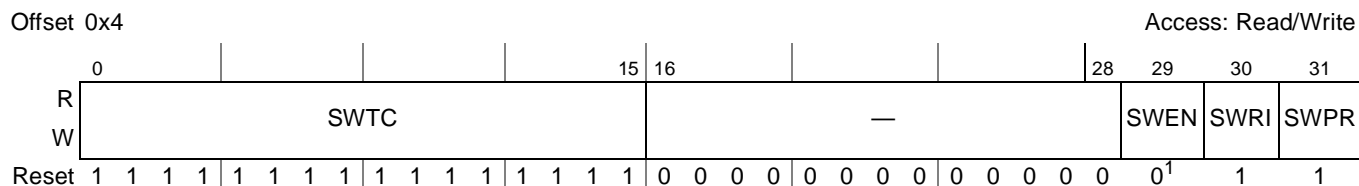
Table 5-33. WDT Register Address Map

Offset	Register	Access	Reset Value	Section/ Page
<b>Watchdog Timer (WDT)—Block Base Address 0x0_0200</b>				
0x000–0x003	Reserved	—	—	—
0x004	System watchdog control register (SWCRR)	R/W	0xFFFF_0003 or 0xFFFF_0007 <sup>1</sup>	5.3.4.1/5-33
0x008	System watchdog count register (SWCNR)	R	0x0000_FFFF	5.3.4.2/5-34
0x00C–0x00D	Reserved	—	—	—
0x00E	System watchdog service register (SWSRR)	R/W	0x0000	5.3.4.3/5-35

<sup>1</sup> SWCRR[SWEN] reset value directly depends on RCWHR[SWEN] (reset configuration word high).

#### 5.3.4.1 System Watchdog Control Register (SWCRR)

The system watchdog control register (SWCRR), shown in Figure 5-22, controls the software watchdog period and configures watchdog timer operation. The SWCRR can be read at any time but can be written only once after system reset.



<sup>1</sup> SWCRR[SWEN] reset value directly depends on RCWHR[SWEN] (reset configuration word high).

Figure 5-22. System Watchdog Control Register (SWCRR)

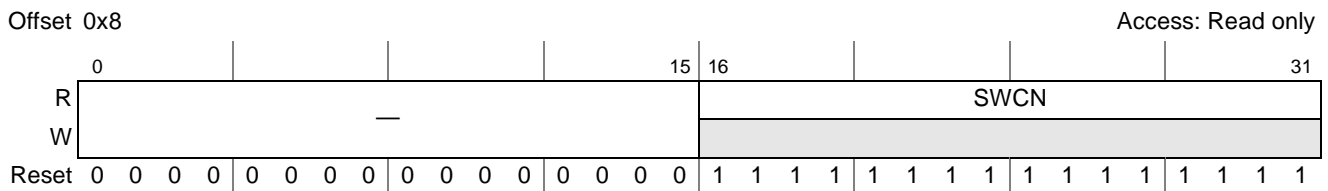
Table 5-34 defines the bit fields of SWCRR.

**Table 5-34. SWCRR Bit Settings**

Bits	Name	Description
0–15	SWTC	Software watchdog time count The SWTC field contains the modulus that is reloaded into the watchdog counter by a service sequence. When a new value is loaded into SWCRR[SWTC], the software watchdog timer is not updated until the servicing sequence is written to the SWSRR. If SWCRR[SWEN] is loaded with 0, the modulus counter does not count. The new value is also used at the next and all subsequent reloads. Reading the SWCRR register returns the value in the system watchdog control register. Reset initializes the SWCRR[SWTC] field to 0xFFFF. <b>Note:</b> The prescaler counter is reset any time a new value is loaded into the watchdog counter and also during reset.
16–28	—	Write reserved, read = 0
29	SWEN	Watchdog enable bit Enables the watchdog timer. The reset value directly depends on the value of the RCWHR[SWEN] bit. It should be cleared by software after a system reset to disable the software watchdog timer. When the watchdog timer is disabled, the watchdog counter and prescaler counter are held in a stopped state. 0 Watchdog timer disabled 1 Watchdog timer enabled <b>Note:</b> After software writes the SWRI bit, the state of SWEN cannot be changed.
30	SWRI	Software watchdog reset/interrupt select bit A WDT timeout causes either a hard reset or machine check interrupt to the core. 0 Software watchdog timer causes a machine check interrupt to the core 1 Software watchdog timer causes a hard reset
31	SWPR	Software watchdog counter prescale bit Controls the divide-by-65,536 WDT counter prescaler 0 The WDT counter is not prescaled. 1 The WDT counter clock is prescaled.

### 5.3.4.2 System Watchdog Count Register (SWCNR)

The system watchdog count register (SWCNR), shown in Figure 5-23, provides visibility to the watchdog counter value. The SWCNR is a read-only register. Writes to SWCNR have no effect and terminate without transfer error exception.



**Figure 5-23. System Watchdog Count Register (SWCNR)**



Table 5-35 defines the bit fields of SWCNR.

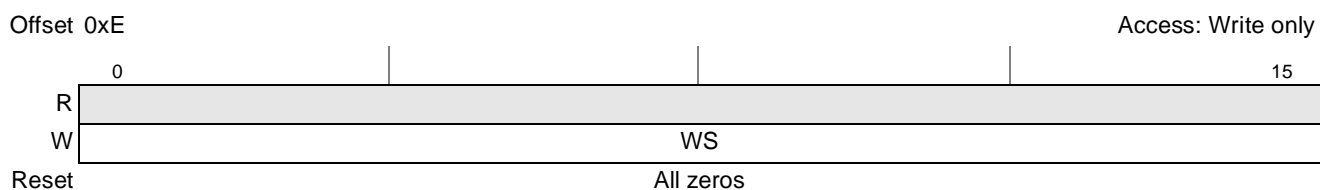
**Table 5-35. SWCNR Bit Settings**

Bits	Name	Description
0–15	—	Write reserved, read = 0
16–31	SWCN	Software watchdog count field. The read-only SWCNR[SWCN] field reflects the current value in the watchdog counter. Writing to the SWCNR register has no effect, and write cycles are terminated normally. Reset initializes the SWCNR[SWCN] field to 0xFFFF. <b>Note:</b> Reading the 16 least-significant bits of 32-bit SWCNR register with two 8-bit reads is not guaranteed to return a coherent value.

### 5.3.4.3 System Watchdog Service Register (SWSRR)

The system watchdog service register (SWSRR) is shown in Figure 5-24. When the watchdog timer is enabled, a write of 0x556C followed by a write 0xAA39 to the SWSRR register before the watchdog counter times out prevents a device reset. If the SWSRR register is not serviced before the timeout, a signal from the watchdog timer to the reset or interrupt controller module asserts a system reset or interrupt (depending on the setting of SWCRR[SWRI]).

Both writes must occur before the timeout in the order listed, but any number of instructions can be executed between the two writes. However, writing any value other than 0x556C or 0xAA39 to the SWSRR register resets the servicing sequence, requiring both values to be written to keep the watchdog timer from causing a reset. Reset initializes the SWSRR[WS] field to 0x0000. SWSRR can be written at any time, but returns all zeros when read.



**Figure 5-24. System Watchdog Service Register (SWSRR)**

Table 5-36 defines the bit fields of SWCNR.

**Table 5-36. SWSRR Bit Settings**

Bits	Name	Description
0–15	WS	Software watchdog service field. The user should periodically write 0x556C followed by 0xAA39 to this register to prevent a software watchdog timer timeout. SWSRR[WS] can be written at any time, but returns all zeros when read.

## 5.3.5 Functional Description

This section provides a functional description of the software watchdog timer (WDT) unit, including a state diagram, block diagram, and discussion of modes of operation.

### 5.3.5.1 Software Watchdog Timer Unit

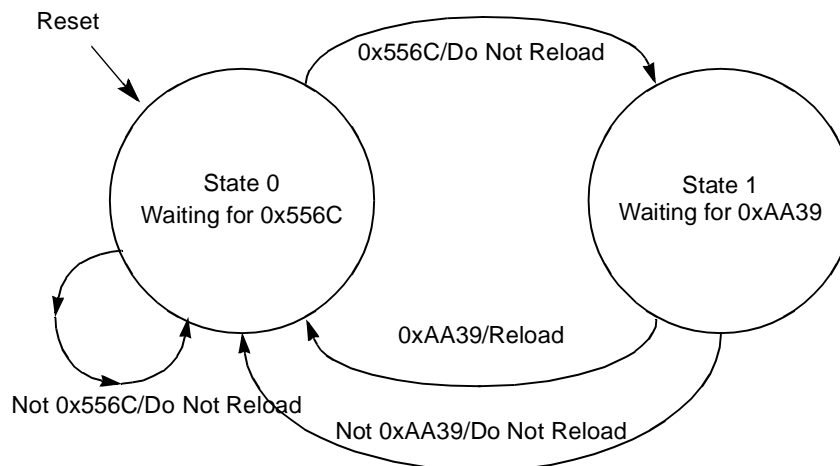
The device provides a software watchdog timer feature to prevent system lock in case the software becomes trapped in loops with no controlled exit. Watchdog timer operations are configured in the system watchdog control register (SWCRR).

The software watchdog timer is enabled after reset to cause a soft reset or non-maskable interrupt (MCP) if it times out. If the software watchdog timer is not needed, the user must clear SWCRR[SWEN] to disable it. If used, the software watchdog timer requires a special service sequence to be executed periodically. Without this periodic servicing, the software watchdog timer times out and issues a reset or a nonmaskable interrupt, as programmed in SWCRR[SWRI]. Once software writes SWRI, the state of SWEN cannot be changed.

The software watchdog timer service sequence consists of the following two steps:

- Write 0x556C to the system watchdog service register (SWSRR)
- Write 0xAA39 to SWSRR

The service sequence reloads the watchdog timer and the timing process begins again. If a value other than 0x556C or 0xAA39 is written to the SWSRR, the entire sequence must start over. Although the writes must occur in the correct order before a time-out, any number of instructions can be executed between the writes. This allows interrupts and exceptions to occur between the two writes when necessary. [Figure 5-25](#) shows a state diagram for the watchdog timer.



**Figure 5-25. Software Watchdog Timer Service State Diagram**

Although most software disciplines permit or even encourage the watchdog concept, some systems require a selection of time-out periods. For this reason, the software watchdog timer must provide a selectable range for the time-out period. Figure 5-26 shows how to handle this need.

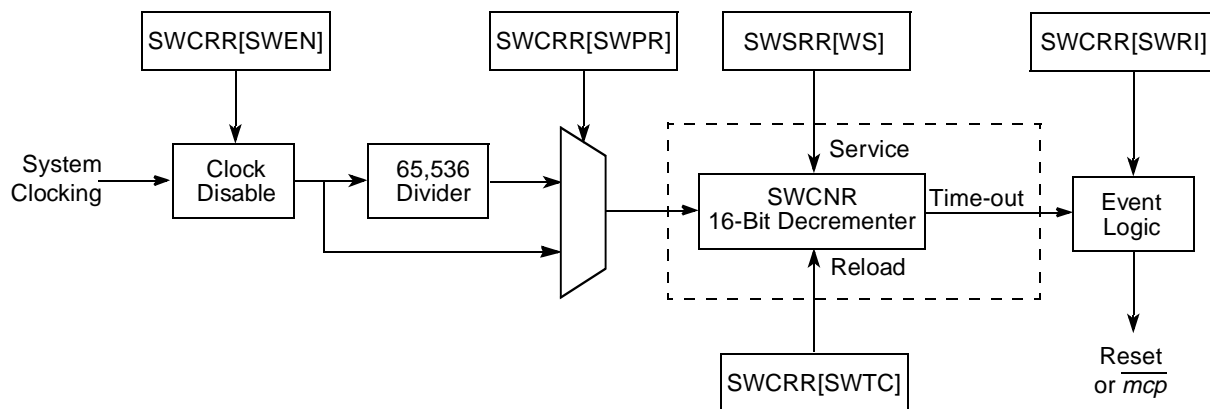


Figure 5-26. Software Watchdog Timer Functional Block Diagram

Figure 5-26 shows that the range is determined by SWCRR[SWTC]. The value in SWTC is then loaded into a 16-bit decremter clocked by the system clock. An additional divide-by-65,536 prescaler value is used when needed.

The decremter begins counting when loaded with a value from SWTC. After the timer reaches 0x0, a software watchdog expiration request is issued to the reset or *mcp* (machine check) control logic. Upon reset, SWTC is set to the maximum value and is again loaded into the system watchdog service register (SWSRR), starting the process over. When a new value is loaded into SWTC, the software watchdog timer is not updated until the servicing sequence is written to the SWSRR. If SWCRR[SWEN] is loaded with 0, the modulus counter does not count.

### 5.3.5.2 Modes of Operation

The WDT unit can operate in the following modes:

- WDT enable/disable mode:

If the software watchdog timer is not needed, the user can disable it. The SWCRR[SWEN] bit enables the watchdog timer. It should be cleared by software after a system reset to disable the software watchdog timer. When the watchdog timer is disabled, the watchdog counter and prescaler counter are held in a stopped state.

— WDT enable mode (SWCRR[SWEN] = 1)

- Default value after hard reset
- Selectable by RCWHR[8].

— WDT disable mode (SWCRR[SWEN] = 0)

- WDT reset/interrupt output mode

Without software periodic servicing, the software watchdog timer times out and issues a reset or a nonmaskable interrupt (*mcp*), programmed in SWCRR[SWRI].

According to the value of SWCRR[SWRI], the WDT timer causes a hard reset or machine check interrupt to the core.

— Reset mode (SWCRR[SWRI] = 1).

Software watchdog timer causes a hard reset (this is the default value after hard reset).

— Interrupt mode (SWCRR[SWRI] = 0).

Software watchdog timer causes a machine check interrupt to the core.

- WDT prescaled/non-prescaled clock mode

The WDT counter clock can be prescaled by programming the SWCRR[SWPR] bit that controls the divide-by-65,536 of the WDT counter.

— Prescale mode (SWCRR[SWPR] = 1)

The WDT clock is prescaled.

— Non-prescale mode (SWCRR[SWPR] = 0)

The WDT clock is not prescaled.

### 5.3.6 Initialization/Application Information (WDT Programming Guidelines)

The software watchdog timer is enabled (by the default value of SWCRR[SWEN]) after reset. The following initialization sequence of WDT is required:

- WDT disabling

If the software watchdog timer is not needed, the user must clear SWCRR[SWEN] bit to disable the WDT not later than its timer times out (~34.36 sec for a 125-MHz system clock).

- WDT initial servicing

If the software watchdog timer is to be used, the special service sequence, described in [Section 5.3.5.1, “Software Watchdog Timer Unit,”](#) must be executed after system reset and not later than the first WDT time-out (~34.36 sec for a 125-MHz system clock).

Subsequently, periodical WDT servicing should be performed according to the programming guidelines given in [Section 5.3.5.1, “Software Watchdog Timer Unit.”](#)

## 5.4 Real Time Clock (RTC) Module

This section describes the theory of operation of the real time clock module including a definition of the external signals and the functions it serves. Additionally, the configuration, control, and status registers are also described.

### 5.4.1 Overview

The device platform provides a real time clock (RTC) timer suitable for time stamping or time and calendar generation. It can maintain a 1-sec count which is unique over a period of approximately 136 years.

The RTC can be initialized by software with an initial count value using the real time counter load register (RTLDR). It can also be programmed to generate an interrupt every second. The real time counter control

register (RTCTR) is used to enable or disable the various timer functions. The real time counter event register (RTEVR) is used to report the interrupt source. The RTC counter is initialized by software and can be disabled if needed.

Figure 5-27 shows the RTC block diagram.

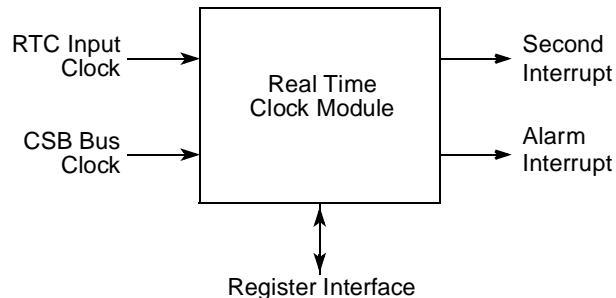


Figure 5-27. RTC Block Diagram

## 5.4.2 Features

The key features of the RTC module include the following:

- 32-bit RTC counter
  - Increments for every one second
  - Can be initialized by software to a specific initial count value
- An alarm function with programmable and maskable alarm interrupt
- Programmable and maskable every second interrupt
- Two possible clock sources
  - External RTC clock—RTC\_PIT\_CLK
  - CSB bus clock
- RTC function can be disabled, if required

## 5.4.3 Assumptions

RTC does not reset during Power On Reset or HRESET and needs to be explicitly reset by software. To reset RTC, set the RTCCR[RESET\_RTC] bit.

## 5.4.4 Modes of operation

The RTC unit can operate in the following modes:

- RTC enable/disable mode
- RTC every-second interrupt enable/disable mode
- RTC alarm interrupt enable/disable mode
- RTC internal/external input clock mode

## 5.4.5 External Signal Description

Table 5-54 lists the external signals for the RTC module.

**Table 5-37. RTC External Signals**

Signal	I/O	Description
RTC_PIT_CLK	I	This signal is used as the timebase for the real time clock module.
		<b>State Meaning</b> —
		<b>Timing</b> 32.768 KHz typical
		<b>Requirements</b> —
		<b>Reset State</b> Always input

## 5.4.6 RTC Memory Map/Register Definition

The RTC programmable register map occupies 32 bytes of memory-mapped space. Reading undefined portions of the memory map returns all zeros; writing has no effect.

All RTC registers are 32 bits wide that are located on 32-bit address boundaries and should only be accessed as 32-bit quantities.

All addresses used in this section are offsets from the RTC base. For more information, see [Chapter 3, “Memory Map.”](#)

Table 5-38 shows the memory map of the RTC.

**Table 5-38. RTC Register Address Map**

Offset	Register	Access	Reset Value <sup>1</sup>	Section/ Page
<b>Real Time Clock (RTC)—Block Base Address 0x0_0300</b>				
0x000	Real time counter control register (RTCNR)	R/W	0x0000_0000	<a href="#">5.4.6.1/5-41</a>
0x004	Real time counter load register (RTLDR)	R/W	0x0000_0000	<a href="#">5.4.6.2/5-42</a>
0x008	Real time counter prescale register (RTPSR)	R/W	0x0000_0000	<a href="#">5.4.6.3/5-42</a>
0x00C	Real time counter register (RTCTR)	R	0x0000_0000	<a href="#">5.4.6.4/5-43</a>
0x010	Real time counter event register (RTEVR)	w1c	0x0000_0000	<a href="#">5.4.6.5/5-43</a>
0x014	Real time counter alarm register (RTALR)	R/W	0xFFFF_FFFF	<a href="#">5.4.6.6/5-44</a>
0x018–0x01F	Reserved	—	—	

<sup>1</sup> It refers to the case where software writes to a specific RTC register, RTCCR. For more information, see [Section 5.4.8, “RTC Reset Sequence.”](#)

### 5.4.6.1 Real Time Counter Control Register (RTCNR)

The real time counter control register (RTCNR), shown in [Figure 5-28](#), is used to enable RTC functions. The register can be read at any time.



**Figure 5-28. Real Time Counter Control Register (RTCNR)**

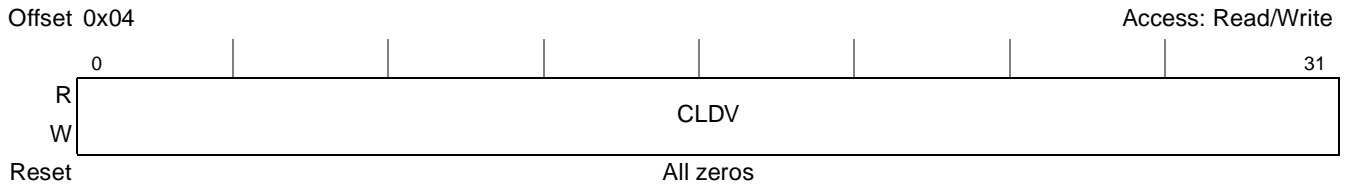
[Table 5-39](#) lists the bit fields of RTCNR.

**Table 5-39. RTCNR Bit Settings**

Bits	Name	Description
0–23	—	Write reserved, read = 0
24	CLEN	Clock enable control bit. This bit controls the counting of the RTC. When the RTC's clock is disabled, the counter maintains its old value. When the counter's clock is enabled, it continues counting using the previous value. 0 Disable counter. 1 Enable counter.
25	CLIN	Input clock control bit. The input clock to the RTC may be either the CSB clock or an external RTC clock. 0 The input clock to the RTC is CSB input clock. 1 The input clock to the RTC is the external RTC clock. If RTC-standby operation is required, this bit should be set to 1.
26–29	—	Write reserved, read = 0
30	AIM	Alarm interrupt mask bit. Used to enable or disable (mask) the RTC alarm interrupt when the RTC's 32-bit counter reaches RTALR[ALR] value. 0 Alarm interrupt generation disabled. 1 Alarm interrupt generation enabled.
31	SIM	Second interrupt mask bit. Used to enable or disable (mask) the RTC periodic interrupt. 0 Periodic interrupt generation disabled. 1 Periodic interrupt generation enabled.

### 5.4.6.2 Real Time Counter Load Register (RTLDR)

The real time counter load register (RTLDR), shown in [Figure 5-29](#), contains the 32-bit value to be loaded in the 32-bit RTC counter.



**Figure 5-29. Real Time Counter Load Register (RTLDR)**

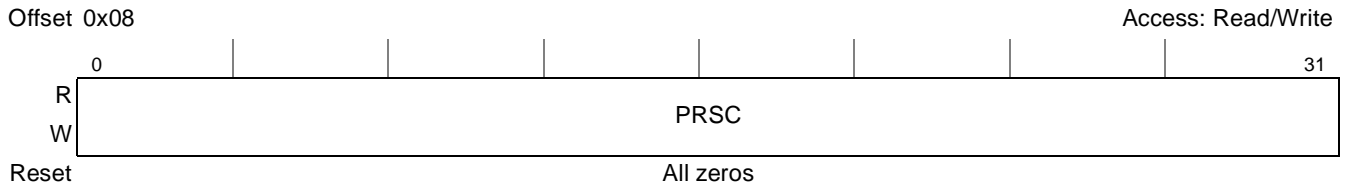
[Table 5-40](#) lists the bit field of RTLDR.

**Table 5-40. RTLDR Bit Settings**

Bits	Name	Description
0–31	CLDV	Contains the 32-bit value to be loaded in the 32-bit RTC counter.

### 5.4.6.3 Real Time Counter Prescale Register (RTPSR)

The real time counter prescale register (RTPSR), shown in [Figure 5-30](#), is a read/write register used to configure the RTC prescaler's value.



**Figure 5-30. Real Time Counter Prescale Register (RTPSR)**

[Table 5-41](#) lists the bit field of RTPSR.

**Table 5-41. RTPSR Bit Settings**

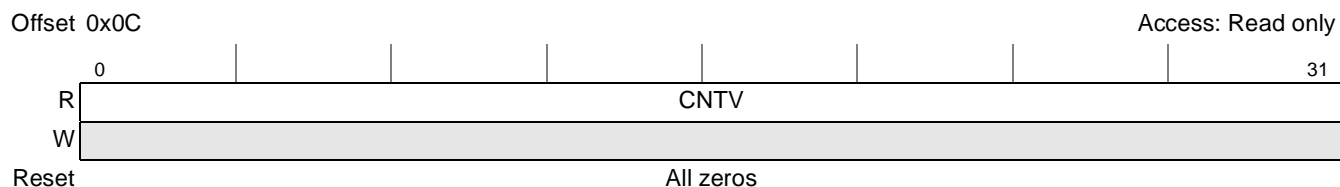
Bits	Name	Description
0–31	PRSC	<p>RTC prescaler bits. Select the input clock divider for the RTC counter clock. The prescaler is programmed to divide the RTC clock input by values from 1 to 4,294,967,296. The value 0x0000 divides the clock by 1 and 0xFFFF_FFFF divides the clock by 4,294,967,296.</p> <p>To accurately predict the timing of the next count, change the RTPSR[PRSC] field only when the enable bit RTCNR[CLE] is clear. Changing the RTPSR[PRSC] bits resets the prescaler counter. RTC reset (for more information, see <a href="#">Section 5.4.8, "RTC Reset Sequence"</a>) and the loading of a new value into the counter both reset the prescaler counter. Clearing RTCNR[CLE] stops the prescaler counter.</p>



### 5.4.6.4 Real Time Counter Register (RTCTR)

The real time counter register (RTCTR), shown in [Figure 5-31](#), is a read-only register that shows the current value in the RTC counter.

The CNTV value is not affected by reads or writes to RTCTR.



**Figure 5-31. Real Time Counter Register (RTCTR)**

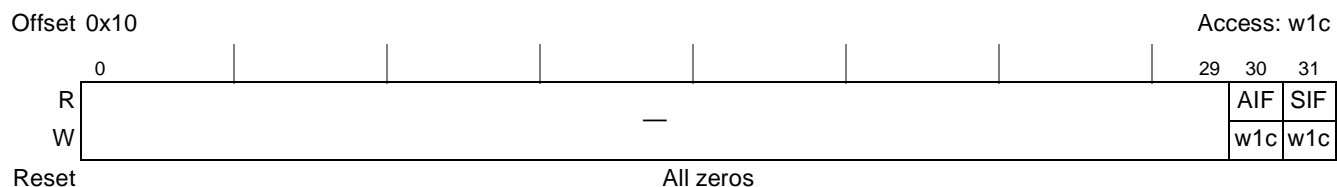
[Table 5-42](#) lists the bit field of RTCTR.

**Table 5-42. RTCTR Bit Settings**

Bits	Name	Description
0–31	CNTV	RTC counter value field. RTCTR[CNTV] contains the current value of the time counter. This is a read-only field. Writes have no effect on RTCTR[CNTV].

### 5.4.6.5 Real Time Counter Event Register (RTEVR)

The real time counter event register (RTEVR), shown in [Figure 5-32](#), is used to report the source of the interrupts. The register can be read at any time.



**Figure 5-32. Real Time Counter Event Register (RTEVR)**

RTEVR bits are cleared by writing ones. Writing zeros does not affect the value of the status bits.

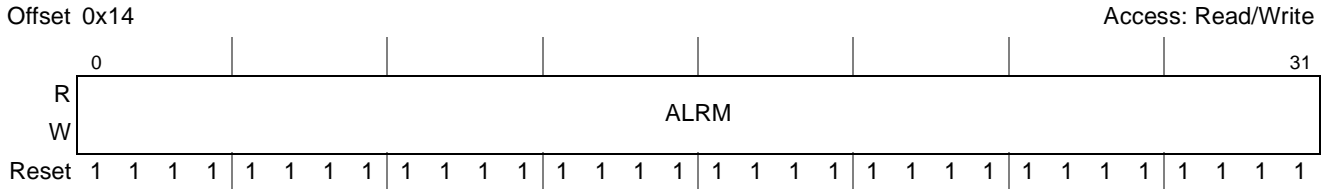
[Table 5-43](#) lists the bit fields of RTEVR.

**Table 5-43. RTEVR Bit Settings**

Bits	Name	Description
0–29	—	Write reserved, read = 0
30	AIF	Alarm interrupt flag bit. Used to indicate the alarm interrupt. It is set if the RTC counter equals RTALR minus one. This bit can be cleared by writing 1.
31	SIF	Second interrupt flag bit. Used to indicate the every-second interrupt. This status bit is set each time that the prescaler count reaches zero. This bit can be cleared by writing 1.

### 5.4.6.6 Real Time Counter Alarm Register (RTALR)

The real time counter alarm register (RTALR), shown in [Figure 5-33](#), contains the 32-bit alarm (ALRM) value. When the value of the RTC counter equals the RTALR[ALRM] value, a maskable interrupt is generated.



**Figure 5-33. Real Time Counter Alarm Register (RTALR)**

[Table 5-44](#) lists the bit field of RTALR.

**Table 5-44. RTALR Bit Settings**

Bits	Name	Description
0–31	ALRM	RTC alarm value. The alarm interrupt is generated when the value of the RTC counter equals RTALR[ALRM].

## 5.4.7 Functional Description

This section provides a functional description of the real time counter unit (RTC), including a block diagram.

### 5.4.7.1 Real Time Counter Unit

The RTC timer is suitable for time stamping or time and calendar generation. It can maintain a one-second count, which is unique over a period of approximately 136 years. Software can convert this count into time-of-day or calendar information, as required. An alarm function is also provided. The RTC can be clocked by the internal system bus clock or by an external clock source. The RTC consists of 32-bit up-counter, which is incremented by a one-second count clock derived from the RTC input clock. The RTC can be programmed to generate a maskable interrupt when the time value matches the value in its associated alarm register.

The RTC can be initialized by software with an initial count value in the real time counter load register (RTLDR). It can also be programmed to generate an interrupt every second. The real time counter control register (RTCTR) is used to enable or disable various timer functions. The real time counter event register (RTEVR) is used to report the interrupt source. The RTC counter is reset to zero on asserting RTCCR[RESET\_RTC]. It is not affected by hard reset or PORESET. The RTC registers are initialized by the software. The RTC function can be disabled by programming the RTC registers.

[Figure 5-34](#) shows the functional RTC block diagram.

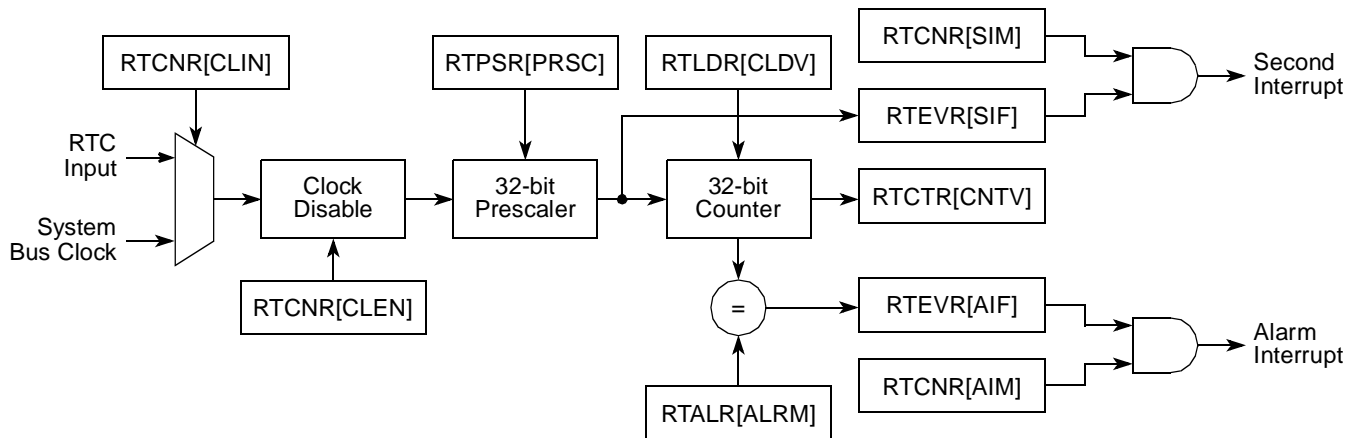


Figure 5-34. Real Time Clock Module Functional Block Diagram

### 5.4.7.2 RTC Operational Modes

The RTC unit can operate in the following modes:

- RTC enable/disable mode
 

RTCNDR[CLEN] enables the RTC timer. It should be set by software, after an RTC reset, to enable the RTC timer.

  - RTC disable mode (RTCNDR[CLEN] = 0)
 

When the RTC's clock is disabled, counter maintains its old value (default).
  - RTC enable mode (RTCNDR[CLEN] = 1)
 

When the counter's clock is enabled, it continues counting using the previous value.
- RTC every-second interrupt enable/disable mode
  - RTC every-second interrupt enable mode (RTCNDR[SIM] = 1)
 

In this mode, RTC sets the RTEVR[SIF] flag and generate an interrupt after the RTC's 32-bit counter reaches zero.
  - RTC every-second interrupt disable mode (RTCNDR[SIM] = 0)
 

In this mode, the RTC sets the RTEVR[SIF] flag but does not generate an interrupt after the RTC's 32-bit counter reaches zero.
- RTC alarm interrupt enable/disable mode
  - RTC alarm interrupt enable mode (RTCNDR[AIM] = 1)
 

In this mode, the RTC sets the RTEVR[AIF] flag and generates an interrupt each time when the RTC's 32-bit counter reaches the RTALR[ALR] value.
  - RTC alarm interrupt disable mode (RTCNDR[AIM] = 0)
 

In this mode, the RTC sets the RTEVR[AIF] flag but does not generate an interrupt when the RTC's 32-bit counter reaches the RTALR[ALR] value.
- RTC internal/external input clock mode
 

The input clock to the RTC may be the CSB clock or an external RTC\_PIT\_CLK.

- RTC uses the internal input clock mode (RTCNR[CLIN] = 0)
- RTC uses the external RTC\_PIT\_CLK (RTCNR[CLIN] = 1)

### 5.4.8 RTC Reset Sequence

The RTC is not reset automatically on POR or HRESET.

- To assert reset to the RTC, program RTCCR[RESET\_RTC] to 1.
- To deassert reset to the RTC, program RTCCR[RESET\_RTC] to 0.

### 5.4.9 RTC Initialization Sequence

The recommended initialization sequence for the RTC is as follows:

1. Write to RTPSR to set the RTC prescaler to the desired value
2. Write to RTLDR to initialize the RTC initial value
3. Write to RTALR to program the RTC alarm value, if needed
4. Write to RTCNR to configure and start the RTC operation: RTC input clock source, second/alarm interrupt mask, and RTC clock enable

## 5.5 Periodic Interval Timer (PIT)

The following sections describe theory of operation of the periodic interval timer (PIT) including a definition of the external signals and the functions it serves. Additionally, the configuration, control, and status registers are described. Note that individual chapters in this reference manual describe additional specific initialization aspects for each individual block.

### 5.5.1 PIT Overview

The periodic interval timer (PIT) that generates periodic interrupts for a real-time operating system or an application software.

The PIT consists of a 32-bit down-counter which is decremented by a clock derived from a CSB clock or from an external 32.768-kHz crystal. The 32-bit counter decrements to zero when loaded with an initial value from the periodic interval timer load register (PTLDR). The periodic interval timer control register (PTCTR) is used to enable or disable the various timer functions. The periodic interval timer event register (PTEVR) is used to report the interrupt source. The PIT function can be disabled if needed.

Figure 5-35 shows the functional PIT block diagram.

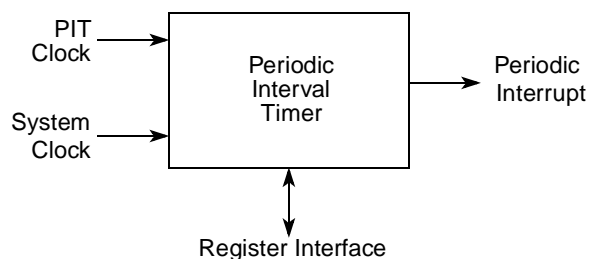


Figure 5-35. Periodic Interval Timer High Level Block Diagram

## 5.5.2 PIT Features

The key features of the PIT include the following:

- Maintains a 32-bit down-counter, clocked by a 32-bit prescaled input clock
- 32-bit PIT counter can be initialized by software to specific initial count value
- Provides programmable and maskable periodic interrupt
- Uses two possible clock sources: the CSB clock or an external RTC\_PIT\_CLOCK.
- PIT function can be disabled

## 5.5.3 PIT Modes of Operation

The PIT unit can operate in the following modes:

- PIT enable/disable mode
- PIT periodic interrupt enable/disable mode
- PIT internal/external input clock mode

## 5.5.4 PIT External Signal Description

This section provides an overview and detailed descriptions of the PIT signals. There is one distinct external input signal (PIT clock), defined in [Table 5-45](#).

Table 5-45. PIT Signal Properties

Name	Function	I/O	Reset	Pull Up
RTC_PIT_CLOCK	Periodic interval timer.	I	N/A	—

[Table 5-46](#) describes of the external PIT signal.

Table 5-46. PIT External Signal—Detailed Signal Descriptions

Signal	I/O	Description
RTC_PIT_CLOCK	I	This signal is used as the timebase for the periodic interval timer module.

## 5.5.5 PIT Memory Map/Register Definition

The PIT programmable register map occupies 32 bytes of memory-mapped space. Reading undefined portions of the memory map returns all zeros; writing has no effect.

All PIT registers are 32 bits wide and reside on 32-bit address boundaries and should only be accessed as 32-bit quantities.

All addresses used in this chapter are offsets from PIT base, as defined in [Chapter 3, “Memory Map.”](#)

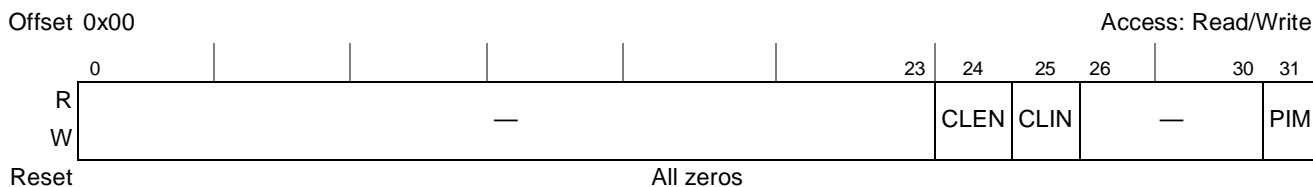
[Table 5-47](#) shows the PIT memory map.

**Table 5-47. PIT Register Address Map**

Offset	Register	Access	Reset Value	Section/ Page
<b>Periodic Interval Timer (PIT)—Block Base Address 0x0_0400</b>				
0x000	Periodic interval timer control register (PTCNR)	R/W	0x0000_0000	<a href="#">5.5.5.1/5-48</a>
0x004	Periodic interval timer load register (PTLDR)	R/W	0x0000_0000	<a href="#">5.5.5.2/5-49</a>
0x008	Periodic interval timer prescale register (PTPSR)	R/W	0x0000_0000	<a href="#">5.5.5.3/5-49</a>
0x00C	Periodic interval timer counter register (PTCTR)	R	0x0000_0000	<a href="#">5.5.5.4/5-50</a>
0x010	Periodic interval timer event register (PTEVR)	w1c	0x0000_0000	<a href="#">5.5.5.5/5-50</a>
0x014–0x01F	Reserved	—	—	

### 5.5.5.1 Periodic Interval Timer Control Register (PTCNR)

The periodic interval timer control register (PTCNR), shown in [Figure 5-36](#), is used to enable the different PIT functions. The register can be read at any time.



**Figure 5-36. Periodic Interval Timer Control Register (PTCNR)**

[Table 5-48](#) defines the bit fields of PTCNR.

**Table 5-48. PTCNR Bit Settings**

Bits	Name	Description
0–23	—	Write reserved, read = 0
24	CLEN	Clock enable control bit. Controls the counting of the PIT. When the PIT’s clock is disabled, the counter maintains its old value. When the counter’s clock is enabled, it continues counting using the previous value. 0 Disable counter. 1 Enable counter.

Table 5-48. PTCNR Bit Settings (continued)

Bits	Name	Description
25	CLIN	Input clock control bit. The input clock to the PIT can be either an internal system clock or an external RTC_PIT_CLOCK. 0 The input clock to the periodic interrupt timer is internal system clock. 1 The input clock to the periodic interrupt timer is external RTC_PIT_CLOCK.
26–30	—	Write reserved, read = 0
31	PIM	Periodic interrupt mask bit. Used to enable or disable (mask) the PIT periodic interrupt. 0 Periodic interrupt generation disabled. 1 Periodic interrupt generation enabled.

### 5.5.5.2 Periodic Interval Timer Load Register (PTLDR)

The periodic interval timer load register (PTLDR), shown in Figure 5-37, contains the 32-bit value to be loaded in a 32-bit PIT counter.

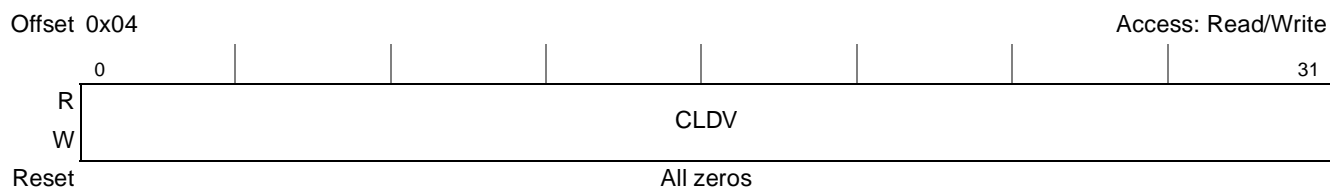


Figure 5-37. Periodic Interval Timer Load Register (PTLDR)

Table 5-49 defines the bit fields of PTLDR.

Table 5-49. PTLDR Bit Settings

Bits	Name	Description
0–31	CLDV	Contains the 32-bit value to be loaded in a 32-bit PIT counter.

### 5.5.5.3 Periodic Interval Timer Prescale Register (PTPSR)

The periodic interval timer prescale register (PTPSR), shown in Figure 5-38, is a read/write register that used to configure the PIT prescaler's value.

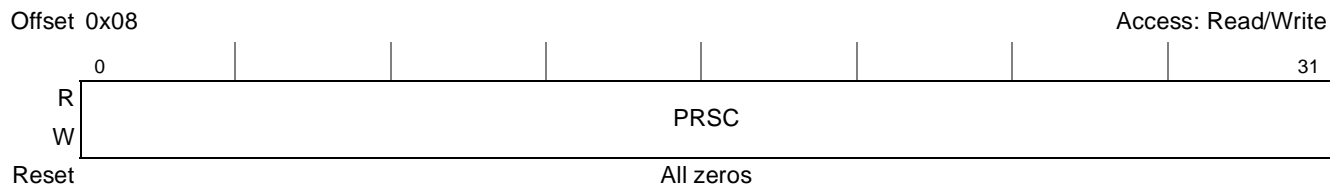


Figure 5-38. Periodic Interval Timer Prescale Register (PTPSR)

Table 5-50 defines the bit fields of PTPSR.

Table 5-50. PTPSR Bit Settings

Bits	Name	Description
0–31	PRSC	PIT prescaler bits. Selects the input clock divider to generate the PIT counter clock. The prescaler is programmed to divide the PIT clock input by values from 1 to 4,294,967,296. The value 0x0000 divides the clock by 1 and 0xFFFF_FFFF divides the clock by 4,294,967,296. To accurately predict the timing of the next count, change the PRSC bit only when the enable bit PTCNR[CLE] is clear. Changing PRSC resets the prescaler counter. System reset and the loading of a new value into the counter also reset the prescaler counter. Clearing the PTCNR[CLE] bit stops the prescaler counter.

#### 5.5.5.4 Periodic Interval Timer Counter Register (PTCTR)

The periodic interval timer counter register (PTCTR), shown in Figure 5-39, is a read-only register that shows the current value in the PIT counter. The PTCTR counter is not affected by reads or writes.

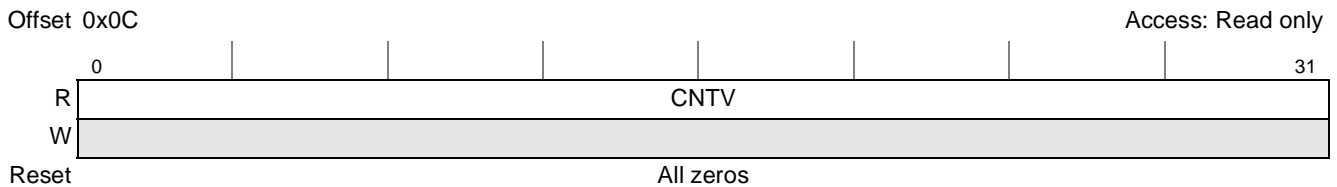


Figure 5-39. Periodic Interval Timer Counter Register (PTCTR)

Table 5-51 defines the bit fields of PTCTR.

Table 5-51. PTCTR Bit Settings

Bits	Name	Description
0–31	CNTV	PIT counter value field. Contains the current value of the time counter. This is a read-only field. Writes have no effect on PTCTR[CNTV].

#### 5.5.5.5 Periodic Interval Timer Event Register (PTEVR)

The periodic interval timer event register (PTEVR), shown in Figure 5-40, is used to report the source of the interrupts. The register can be read at any time. PTEVR bits are cleared by writing ones. Writing zeros does not affect the value of the status bits.

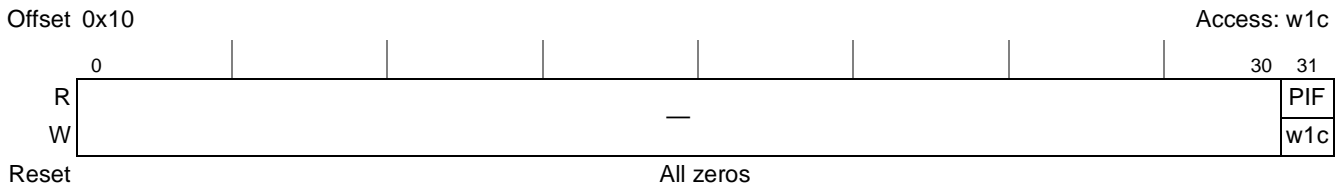


Figure 5-40. Periodic Interval Timer Event Register (PTEVR)



Table 5-52 defines the bit fields of PTEVR.

Table 5-52. PTEVR Bit Settings

Bits	Name	Description
0–30	—	Write reserved, read = 0
31	PIF	Periodic interrupt flag bit. Used to indicate the periodic interrupt. It is asserted after the SPMPIT counter counts to zero. If PTCNR[PIM] bit is enabled, then an interrupt would be generated. This status bit should be cleared by software.

## 5.5.6 Functional Description

This section provides a functional description of the PIT unit, including a block diagram and discussion of operational modes.

### 5.5.6.1 Periodic Interval Timer Unit

The PIT generates periodic interrupts for use with a real-time operating system or the application software. It consists of a 32-bit down-counter which is decremented by a clock derived from the CSB clock or from the RTC\_PIT\_CLOCK. The 32-bit counter decrements to zero when loaded with a initial value from the periodic interval timer load register (PTLDR). After the timer reaches zero, PTEVR[PIF] is set and an interrupt is generated if PTCNR[PIM] = 1. At the next count cycle, the value in the PTLDR[CLDV] is loaded into the counter and the process repeats. When a new value is loaded into the PTLDR[CLDV], the PIT is updated, the prescaler counter is reset, and the counter begins counting. Setting of PTEVR[PIF] generates an interrupt that remains pending until PTEVR[PIF] is cleared. If PTEVR[PIF] is set again before being cleared, the interrupt remains pending until PTEVR[PIF] is cleared. Any write to the PTLDR[CLDV] stops the current countdown and the count resumes with the new value in PTLDR[CLDV]. If PTCNR[CLEN] = 0, the PIT cannot count and retains the old count value. PTCTR contain the PIT current value. The PIT function can be disabled if needed.

Figure 5-41 shows the functional PIT block diagram.

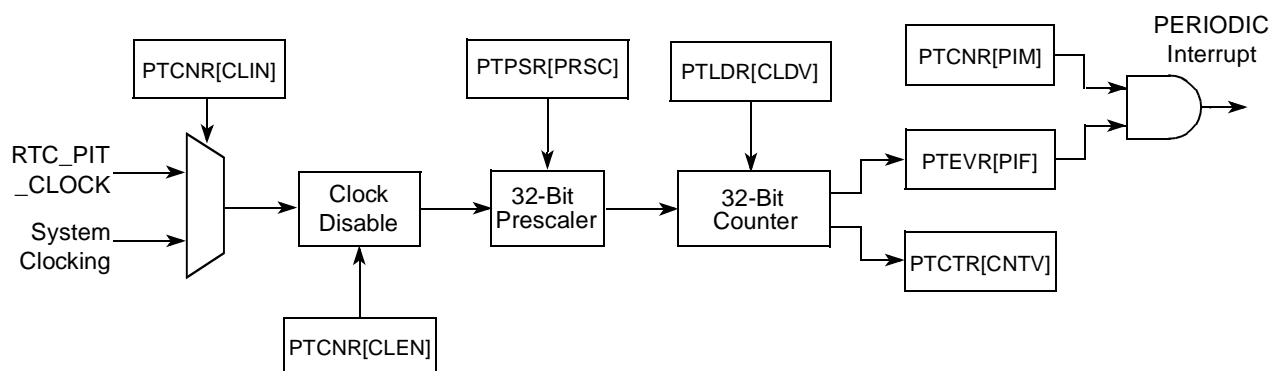


Figure 5-41. Periodic Interval Timer Functional Block Diagram

### 5.5.6.2 PIT Operational Modes

The PIT unit can operate in the following modes:

- PIT enable/disable mode:
 

The PTCNR[CLEN] bit enables the PIT timer. It should be set by software after a system reset to enable the PIT timer.

  - PIT disable mode (PTCNR[CLEN] = 0). When the PIT's clock is disabled, counter maintains its old value.
  - PIT enable mode (PTCNR[CLEN] = 1). When the counter's clock is enabled, it continues counting using the previous value.
- PIT periodic interrupt enable/disable mode:
  - PIT periodic interrupt enable mode (PTCNR[PIM] = 1). After the PIT's 32-bit counter reaches zero, the PIT sets the PTEVR[PIF] flag and generates an interrupt.
  - PIT periodic interrupt disable mode (PTCNR[PIM] = 0). After the PIT's 32-bit counter reaches zero, the PIT sets the PTEVR[PIF] flag but does not generate an interrupt.
- PIT internal/external input clock mode:
 

The input clock to the PIT may be an internal system clock or the RTC\_PIT\_CLOCK.

  - PIT use the internal input clock mode (PTCNR[CLIN] = 0)
  - PIT use the RTC\_PIT\_CLOCK (PTCNR[CLIN] = 1)

### 5.5.7 PIT Programming Guidelines

The following initialization sequence of PIT is recommended:

1. Write to PTPSR to set the PIT prescaler to the desired value
2. Write to PTLDR to initialize the PIT initial value
3. Write to PTCNR to configure and start the PIT operation: PIT input clock source, periodic interrupt mask, PIT clock enable.

## 5.6 General-Purpose Timers (GTM)

The following sections describe theory of operation of the general purpose (global) timer module, including a definition of the external signals and the functionality. Additionally, the configuration, control, and status registers are described. Note that individual chapters in this book describe additional specific initialization aspects for each individual block.

There is one global timer (GTM1) with 4 instances as GTM1\_1, GTM1\_2, GTM1\_3, and GTM1\_4.

### 5.6.1 GTM Overview

Each global timer module includes four identical 16-bit general-purpose timers, two 32-bit timers or one 64-bit timer. Each GTM timer consists of a timer prescale register (GTPSR), a timer mode register (GTMDR) a timer capture register (GTCPR), a timer counter register (GTCNR), a timer reference register

(GTRFR), a timer event register (GTEVR), and a timer global configuration register (GTCFR). The GTPSRs and the GTMDRs contain the primary and secondary prescalers, programmed by the user.

Figure 5-42 shows the functional GTM block diagram.

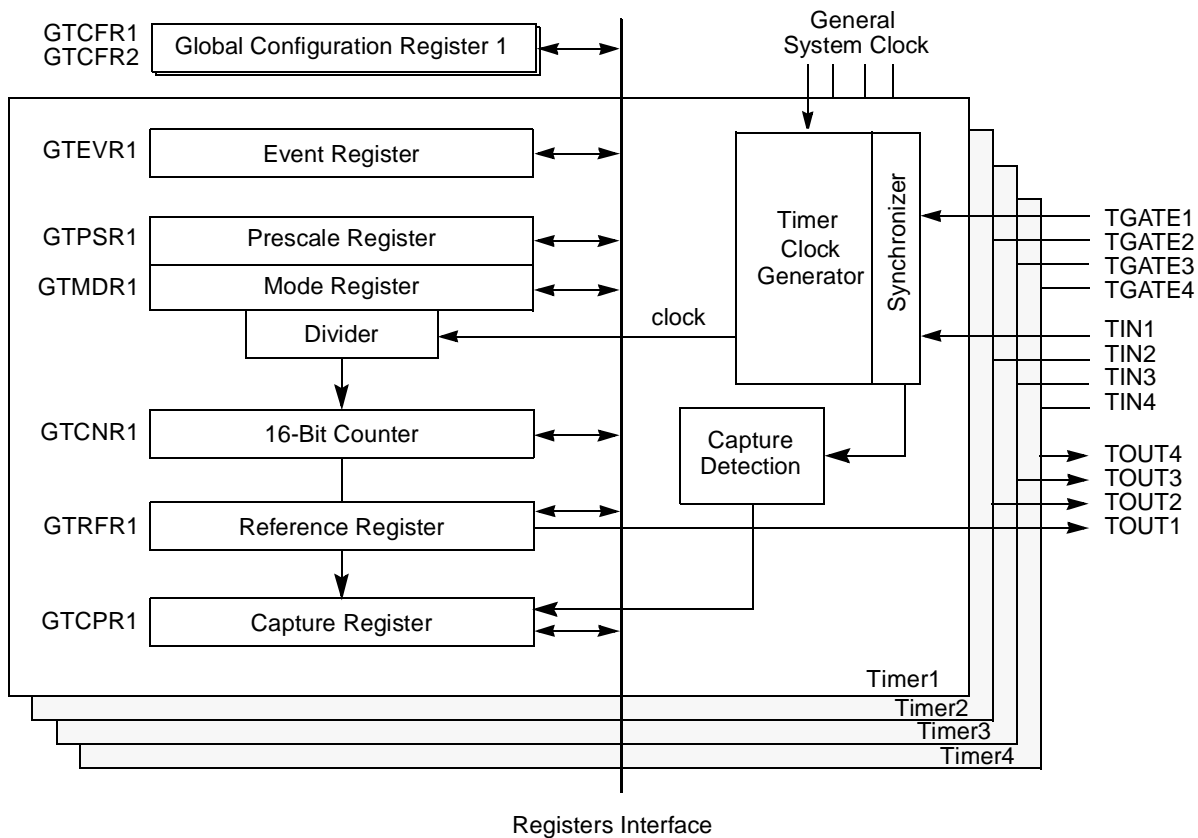


Figure 5-42. Global Timers Block Diagram

## 5.6.2 GTM Features

The key features of the timer include the following:

- The maximum input clock is the system bus clock
- Four 16-bit programmable timers
- Two timers cascaded internally or externally (using TIN and TOUT external signals) to form a 32-bit timer
- One timer cascaded internally or externally to form a 64-bit timer
- Maximum period of ~549.6 second (at 125-MHz bus clock and two prescalers each = 256 and slow go mode = 4) for 16-bit timer
- Maximum period of ~8796 seconds (at 125-MHz bus clock and a prescaler = 256) for 32-bit timer
- Maximum period of thousands of years (at 125-MHz bus clock and prescaler = 256) for 64-bit timer
- 8-nanosecond timer resolution (at 125-MHz bus clock and no prescaler)

- Resolution and maximum period can be traded off by selecting prescaler divisor
- Three programmable input clock sources for the timer prescalers
- Input capture capability
- Output compare with programmable mode for the output pin
- Free run and restart modes
- Functional and programming compatibility with MPC8260 timers

### 5.6.3 GTM Modes of Operation

The GTM unit can operate in the following modes:

- Cascaded modes
- Clock source modes
- Reference modes
- Capture modes

#### 5.6.3.1 Cascaded Modes

GTCFR $n$ [PCAS] and GTCFR2[SCAS] are used to put the timers into different cascaded modes:

- Non-cascaded mode: Each timer (timer 1, timer 2, timer 3, and timer 4), function as a independent 16-bit timer with a 16-bit GTRFR, GTCPR, GTMDR, and GTCNR. In this mode, the non-cascaded GTRFR, GTCPR, and GTCNR should be referenced with corresponding 16-bit bus cycles.
- Pair-cascaded mode: In this mode, two 16-bit timers can be internally cascaded to form a 32-bit counter: timer 1 can be internally cascaded to timer 2 and timer 3 may be internally cascaded to timer 4. Because the decision to cascade timers is made independently, the user has the option of selecting two 16-bit timers and one 32-bit timer, or two 32-bit timers. When working in the pair-cascaded mode, the cascaded GTRFR, GTCPR, and GTCNR should be referenced with 32-bit bus cycles.
- Super-cascaded mode: In this mode, all four 16-bit timers can be internally cascaded to form a 64-bit counter. When working in the super-cascaded mode, the cascaded GTRFR, GTCPR, and GTCNR should be referenced with two 32-bit bus cycles.

#### 5.6.3.2 Clock Source Modes

The clock input to the timer's prescaler can be selected from three sources:

- The system clock
- The system slow go clock (system bus clock internally divided by 16)
- The corresponding TIN $x$  pin

### 5.6.3.3 Reference Modes

Each timer can be configured to count until a reference is reached and then either begin a new time count immediately or continue to run. The FRR bit of the corresponding GTMRR selects each mode.

- Free run reference mode. The corresponding timer count continues to increment after the reference value is reached.
- Reset reference mode. The corresponding timer count is reset immediately after the reference value is reached.

### 5.6.3.4 Capture Modes

Each timer has a 16-bit field in GTCPR, used to latch the value of the counter when a defined transition of TIN<sub>x</sub> is sensed by the corresponding input capture edge detector.

- Normal gate mode enables the count on a falling edge of the  $\overline{\text{TGATE}}$  pin and disables the count on the rising edge of  $\overline{\text{TGATE}}$ . This mode allows the timer to count conditionally, based on the state of  $\overline{\text{TGATE}}$ .
- The restart gate mode performs the same function as normal mode, except it also resets the counter on the falling edge of the  $\overline{\text{TGATE}}$  pin. This mode has applications in pulse interval measurement and bus monitoring.

## 5.6.4 GTM External Signal Description

This section provides an overview and detailed descriptions of the GTM signals.

There are four distinct external input timer capture signals (TIN1, TIN2, TIN3, and TIN4), four distinct external input timer gate signals ( $\overline{\text{TGATE1}}$ ,  $\overline{\text{TGATE2}}$ ,  $\overline{\text{TGATE3}}$ , and  $\overline{\text{TGATE4}}$ ), and four distinct external timer output signals ( $\overline{\text{TOUT1}}$ ,  $\overline{\text{TOUT2}}$ ,  $\overline{\text{TOUT3}}$ , and  $\overline{\text{TOUT4}}$ ). The GTM interface signals are defined in [Table 5-53](#).

**Table 5-53. GTM Signal Properties**

Name	Port	Function	I/O	Reset	Require Pull Up
TIN1	TIN1	Global timer 1 capture control signal	I	0	No
TIN2	TIN2	Global timer 2 capture control signal	I	0	No
TIN3	TIN3	Global timer 3 capture control signal	I	0	No
TIN4	TIN4	Global timer 4 capture control signal	I	0	No
$\overline{\text{TGATE1}}$	$\overline{\text{TGATE1}}$	Global timer 1 counter gate control signal	I	0	No
$\overline{\text{TGATE2}}$	$\overline{\text{TGATE2}}$	Global timer 2 counter gate control signal	I	0	No
$\overline{\text{TGATE3}}$	$\overline{\text{TGATE3}}$	Global timer 3 counter gate control signal	I	0	No
$\overline{\text{TGATE4}}$	$\overline{\text{TGATE4}}$	Global timer 4 counter gate control signal	I	0	No
$\overline{\text{TOUT1}}$	$\overline{\text{TOUT1}}$	Global timer 1 counter output signal	O	1	No
$\overline{\text{TOUT2}}$	$\overline{\text{TOUT2}}$	Global timer 2 counter output signal	O	1	No

Table 5-53. GTM Signal Properties (continued)

Name	Port	Function	I/O	Reset	Require Pull Up
$\overline{\text{TOUT3}}$	$\overline{\text{TOUT3}}$	Global timer 3 counter output signal	O	1	No
$\overline{\text{TOUT4}}$	$\overline{\text{TOUT4}}$	Global timer 4 counter output signal	O	1	No

Table 5-54 provides detailed descriptions of the external GTM signals.

Table 5-54. GTM External Signals—Detailed Signal Descriptions

Signal	I/O	Description
$\text{TIN}_n$	I	Global timer capture control signal. Used to latch the value of the counter when a defined transition of $\text{TIN}_n$ is sensed by the corresponding input capture edge detector.
		<b>State Meaning</b> Asserted/Negated—According to the programmed polarity by the corresponding $\text{GTMDR}_n[\text{CE}]$ . Each timer has a 16-bit $\text{GTCPR}$ used to latch the value of the counter when a defined transition of $\text{TIN}_n$ is sensed by the corresponding input capture edge detector. Upon a capture or reference event, the corresponding $\text{GTEVR}$ bit is set and a maskable interrupt request is issued to the interrupt controller.
		<b>Timing</b> Assertion/Negation—Asynchronous to internal bus clock. $\text{TIN}_n$ is internally synchronized to the system bus clock. If $\text{TIN}_n$ meets the asynchronous input setup time, the value of counter is captured after one system bus clock when working with the internal clock.
$\overline{\text{TGATE}}_n$	I	Global timer counter gate control signal. Used to gate/restart the counter when a defined transition of $\overline{\text{TGATE}}_n$ is sensed by the corresponding input capture edge detector.
		<b>State Meaning</b> Asserted/Negated—According to the programmed polarity by the corresponding $\text{GTCFR}[\text{GM}_x]$ bits. In a restart gate mode ( $\text{GTCFR}[\text{GM}_n] = 0$ ), the $\overline{\text{TGATE}}_n$ pin is used to enable/disable count. A falling $\overline{\text{TGATE}}_n$ pin enables and restarts the count and a rising edge of $\overline{\text{TGATE}}_n$ disables the count. In a normal gate mode ( $\text{GTCFR}[\text{GM}_n] = 1$ ), the $\overline{\text{TGATE}}_n$ have similar functionality, except the falling edge of $\overline{\text{TGATE}}_n$ does not restart the appropriate count value in $\text{GTCNR}_n[\text{CNV}_n]$ .
		<b>Timing</b> Assertion/Negation—Asynchronous to internal bus clock. $\overline{\text{TGATE}}_n$ is internally synchronized to the system bus clock. If $\overline{\text{TGATE}}_n$ meets the asynchronous input setup time, the counter begins counting or stop counting depending on the signal state and configured mode.
$\overline{\text{TOUT}}_n$	O	Global timer counter output signal. The GTM output a signal on the timer output pin $\overline{\text{TOUT}}_n$ when the reference value is reached.
		<b>State Meaning</b> Asserted/Negated—According to the programmed polarity by the corresponding $\text{GTMDR}_n[\text{OM}_n]$ . <ol style="list-style-type: none"> <li>Active-low pulse on <math>\overline{\text{TOUT}}_n</math> for one timer input clock cycle as defined by the <math>\text{GTMDR}_n[\text{CLK}_n]</math> bits (<math>\text{GTMDR}_n[\text{OM}_n] = 1</math>). Thus, <math>\overline{\text{TOUT}}_n</math> may be low for one general system clock period, one general system slow go clock period, or one <math>\text{TIN}_n</math> pin clock cycle period.</li> <li>Toggle the <math>\overline{\text{TOUT}}_n</math> pin (<math>\text{GTMDR}_n[\text{OM}_n] = 0</math>). <math>\overline{\text{TOUT}}_n</math> changes occur on the rising edge of the timer input clock.</li> </ol>
		<b>Timing</b> Assertion/Negation— $\overline{\text{TOUT}}_n$ changes occur on the rising edge of the timer input clock.

## 5.6.5 GTM Memory Map/Register Definition

The GTM programmable register map occupies 64 bytes of memory-mapped space. Reading undefined portions of the memory map returns all zeros; writing has no effect.

All GTM registers are 8 or 16 bits wide, located on 8-bit or 16-bit address boundaries, and should only be accessed as 8-bit or 16-bit quantities. All addresses used in this chapter are offsets from GTM, as defined in [Chapter 3, “Memory Map.”](#)

[Table 5-55](#) shows the memory map of the GTM.

**Table 5-55. GTM Register Address Map**

Offset	Register	Access	Reset Value	Section/ Page
<b>General Purpose (Global) Timer Module 1—Block Base Address 0x0_0500</b>				
0x000	Timer 1 and 2 global timers configuration register (GTCFR1)	R/W	0x0000	<a href="#">5.6.5.1/5-58</a>
0x001–0x003	Reserved	—	—	—
0x004	Timer 3 and 4 global timers configuration register (GTCFR2)	R/W	0x0000	<a href="#">5.6.5.1/5-58</a>
0x005–0x00F	Reserved	—	—	—
0x010	Timer 1 global timers mode register (GTMDR1)	R/W	0x0000	<a href="#">5.6.5.2/5-61</a>
0x012	Timer 2 global timers mode register (GTMDR2)			
0x014	Timer 1 global timers reference register (GTRFR1)	R/W	0xFFFF	<a href="#">5.6.5.3/5-62</a>
0x016	Timer 2 global timers reference register (GTRFR2)			
0x018	Timer 1 global timers capture register (GTCPR1)	R/W	0x0000	<a href="#">5.6.5.4/5-62</a>
0x01A	Timer 2 global timers capture register (GTCPR2)			
0x01C	Timer 1 global timers counter register (GTCNR1)	R/W	0x0000	<a href="#">5.6.5.5/5-63</a>
0x01E	Timer 2 global timers counter register (GTCNR2)			
0x020	Timer 3 global timers mode register (GTMDR3)	R/W	0x0000	<a href="#">5.6.5.2/5-61</a>
0x022	Timer 4 global timers mode register (GTMDR4)			
0x024	Timer 3 global timers reference register (GTRFR3)	R/W	0xFFFF	<a href="#">5.6.5.3/5-62</a>
0x026	Timer 4 global timers reference register (GTRFR4)			
0x028	Timer 3 global timers capture register (GTCPR3)	R	0x0000	<a href="#">5.6.5.4/5-62</a>
0x02A	Timer 4 global timers capture register (GTCPR4)			
0x02C	Timer 3 global timers counter register (GTCNR3)	R/W	0x0000	<a href="#">5.6.5.5/5-63</a>
0x02E	Timer 4 global timers counter register (GTCNR4)			
0x030	Timer 1 global timers event register (GTEVR1)	w1c	0x0000	<a href="#">5.6.5.6/5-63</a>
0x032	Timer 2 global timers event register (GTEVR2)			
0x034	Timer 3 global timers event register (GTEVR3)			
0x036	Timer 4 global timers event register (GTEVR4)			

Table 5-55. GTM Register Address Map (continued)

Offset	Register	Access	Reset Value	Section/ Page
0x038	Timer 1 global timers prescale register (GTPSR1)	R/W	0x0003	5.6.5.7/5-64
0x03A	Timer 2 global timers prescale register (GTPSR2)			
0x03C	Timer 3 global timers prescale register (GTPSR3)			
0x03E	Timer 4 global timers prescale register (GTPSR4)			

### 5.6.5.1 Global Timers Configuration Registers (GTCFR $n$ )

The global timers configuration registers (GTCFR1 and GTCFR2), shown in [Figure 5-43](#) and [Figure 5-44](#), contain configuration parameters used by the timers. These registers allow simultaneous starting, stopping and resetting of a pair of timers (1 and 2 or 3 and 4) or of a groups of timers (1, 2, 3, and 4) if one bus cycle is used. GTCFR is cleared by reset.

#### NOTE

For proper operation of the timers, do not change the modes of operation and enable the timer in the same register write operation. The modes can be changed when GTCFR $n$ [RST $n$ ] is cleared. However, when GTCFR $n$ [RST $n$ ] are set, they are the only bits that can be changed.

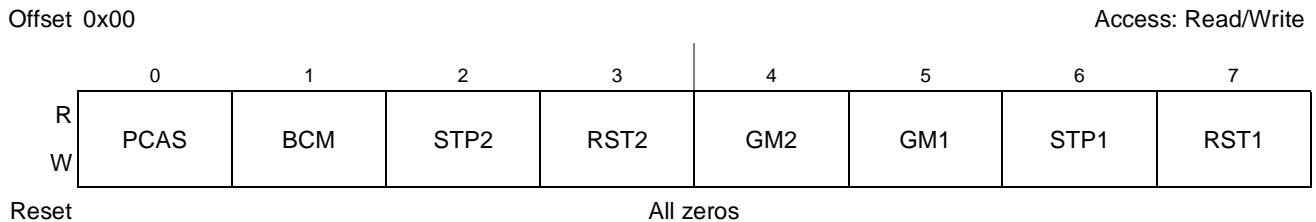


Figure 5-43. Global Timers Configuration Register 1 (GTCFR1)

[Table 5-56](#) defines the bit fields of GTCFR1.

Table 5-56. GTCFR1 Bit Settings

Bits	Name	Description
0	PCAS	Pair-cascade mode 0 Normal operation 1 Timers 1 and 2 cascade to form a 32-bit timer. <b>Note:</b> This bit is ignored in super-cascade mode (GTCFR2[SCAS] = 1). <b>Note:</b> It is allowed to change the value of this bit only when the corresponding timers are in reset mode. Thus, the user should first clear the RST1 and RST2 bits (without changing PCAS) and then, in a separate write to the register, change the value of PCAS.
1	BCM	Backward-compatible mode 0 Provide backward compatibility to PowerQUICC II family timers. In this mode GTCFR1[GM2] bit controls the gate mode for timers 1 and 2 and GTCFR2[GM4] bit controls the gate mode for timers 3 and 4. GTCFR1[GM1] and GTCFR2[GM3] bits are ignored. 1 Normal operational mode



Table 5-56. GTCFR1 Bit Settings (continued)

Bits	Name	Description
2	STP2	Stop timer 2 0 Normal operation 1 Reduce power consumption of the corresponding timer. This bit stops all clocks to the timer 2, except the Register Interface clock, which allows to read and write timer registers. The clocks to the timer remain stopped until the user clears this bit or a hardware reset occurs.
3	RST2	Reset timer 2 0 Reset the timer 2, including GTMDR2, GTRFR2, GTCNR2, GTCPR2, and GTEVR2 (a software reset is identical to an external reset). 1 Enable the corresponding timer if the STP2 bit is cleared.
4	GM2	Gate mode for $\overline{\text{TGATE2}}$ 0 Restart gate mode. The $\overline{\text{TGATE2}}$ pin is used to enable/disable count. A low level of $\overline{\text{TGATE2}}$ enables and a falling edge of $\overline{\text{TGATE2}}$ restarts the count (reset the dynamic counter's count value to 0) and a high level of $\overline{\text{TGATE2}}$ disables the count. 1 Normal gate mode. This mode is the same as 0, except the falling edge of $\overline{\text{TGATE2}}$ does not restart the appropriate count value in GTCNR2[CNV2].
5	GM1	Gate mode for $\overline{\text{TGATE1}}$ 0 Restart gate mode. The $\overline{\text{TGATE1}}$ is used to enable/disable count. A low level of $\overline{\text{TGATE1}}$ enables and a falling edge of $\overline{\text{TGATE1}}$ restarts the count (reset the dynamic counter's count value to 0) and a high level of $\overline{\text{TGATE1}}$ disables the count. 1 Normal gate mode. This mode is the same as 0, except the falling edge of $\overline{\text{TGATE1}}$ does not restart the appropriate count value in GTCNR1[CNV1]. <b>Note:</b> In backward-compatible mode (GTCFR1[BCM] = 0) this bit is ignored. GTCFR1[GM2] bit controls the gate mode for timers 1 and 2.
6	STP1	Stop timer 1 0 Normal operation 1 Reduce power consumption of the corresponding timer. This bit stops all clocks to the timer 1, except the register interface clock, which allows to read and write timer registers. The clocks to the timer remain stopped until the user clears this bit or a hardware reset occurs.
7	RST1	Reset timer 1 0 Reset the timer 1, including GTMDR1, GTRFR1, GTCNR1, GTCPR1, and GTEVR1 (a software reset is identical to an external reset). 1 Enable the corresponding timer if the STP1 bit is cleared.

The GTCFR2 register is shown in [Figure 5-44](#).

Offset 0x04

Access: Read/Write

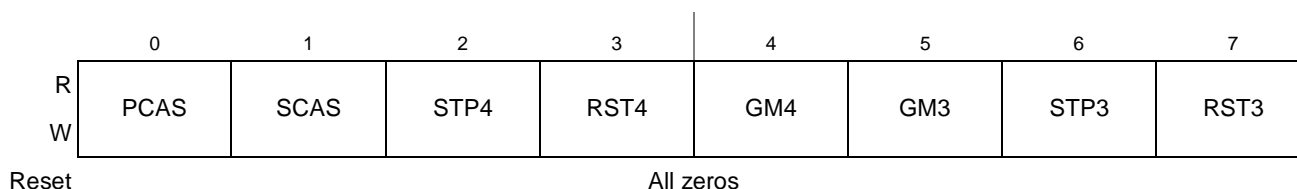


Figure 5-44. Global Timers Configuration Register 2 (GTCFR2)

Table 5-57 defines the bit fields of GTCFR2.

**Table 5-57. GTCFR2 Bit Settings**

Bits	Name	Description
0	PCAS	<p>Pair-cascade mode</p> <p>0 Normal operation.</p> <p>1 Timers 3 and 4 cascade to form a 32-bit timer.</p> <p><b>Note:</b> This bit is ignored in super-cascade mode (GTCFR2[SCAS] = 1).</p> <p><b>Note:</b> It is allowed to change the value of this bit only when the corresponding timers are in reset mode. Thus, the user should first clear the RST3 and RST4 bits (without changing PCAS) and then, in a separate write to the register, change the value of PCAS.</p>
1	SCAS	<p>Super cascade mode</p> <p>0 Normal operation</p> <p>1 Timers 1, 2, 3 and 4 cascade to form a 64-bit timer.</p> <p><b>Note:</b> In super-cascade mode (GTCFR2[SCAS] = 1) the pair-cascade mode bits are ignored, (GTCFR1/2[PCAS] = Don't Care).</p> <p><b>Note:</b> It is allowed to change the value of this bit only when the corresponding timers are in reset mode. Thus, the user should first clear the RST1, RST2, RST3, and RST4 bits (without changing SCAS) and then, in a separate write to the register, change the value of SCAS.</p>
2	STP4	<p>Stop timer 4</p> <p>0 Normal operation</p> <p>1 Reduce power consumption of the corresponding timer. This bit stops all clocks to the timer 4, except the register interface clock, which allows to read and write timer registers. The clocks to the timer remain stopped until the user clears this bit or a hardware reset occurs.</p>
3	RST4	<p>Reset timer 4</p> <p>0 Reset the timer 4, including GTMDR4, GTRFR4, GTCNR4, GTCPR4, and GTEVR4 (a software reset is identical to an external reset).</p> <p>1 Enable the corresponding timer if the STP4 bit is cleared.</p>
4	GM4	<p>Gate mode for <math>\overline{\text{TGATE4}}</math></p> <p>0 Restart gate mode. The <math>\overline{\text{TGATE4}}</math> is used to enable/disable count. A low level of <math>\overline{\text{TGATE4}}</math> enables and a falling edge of <math>\overline{\text{TGATE4}}</math> restarts the count (reset the dynamic counter's count value to 0) and a high level of <math>\overline{\text{TGATE4}}</math> disables the count.</p> <p>1 Normal gate mode. This mode is the same as 0, except the falling edge of <math>\overline{\text{TGATE4}}</math> does not restart the appropriate count value in GTCNR4[CNV4].</p>
5	GM3	<p>Gate mode for <math>\overline{\text{TGATE3}}</math></p> <p>0 Restart gate mode. The <math>\overline{\text{TGATE3}}</math> is used to enable/disable count. A low level of <math>\overline{\text{TGATE3}}</math> enables and a falling edge of <math>\overline{\text{TGATE3}}</math> restarts the count (reset the dynamic counter's count value to 0) and a high level of <math>\overline{\text{TGATE3}}</math> disables the count.</p> <p>1 Normal gate mode. This mode is the same as 0, except the falling edge of <math>\overline{\text{TGATE3}}</math> does not restart the appropriate count value in GTCNR3[CNV3].</p> <p><b>Note:</b> In backward-compatible mode (GTCFR1[BCM] = 0) this bit is ignored. The GTCFR2[GM4] bit controls the gate mode for timers 3 and 4.</p>
6	STP3	<p>Stop timer 3</p> <p>0 Normal operation</p> <p>1 Reduce power consumption of the corresponding timer. This bit stops all clocks to the timer 3, except the register interface clock, which allows to read and write timer registers. The clocks to the timer remain stopped until the user clears this bit or a hardware reset occurs.</p>
7	RST3	<p>Reset timer 3</p> <p>0 Reset the timer 3, including GTMDR3, GTRFR3, GTCNR3, GTCPR3, and GTEVR3 (a software reset is identical to an external reset).</p> <p>1 Enable the corresponding timer if the STP3 bit is cleared.</p>

### 5.6.5.2 Global Timers Mode Registers (GTMDR1–GTMDR4)

The global timers mode registers (GTMDR1, GTMDR2, GTMDR3, and GTMDR4) are shown in Figure 5-45.

Erratic behavior may occur if GTCFR1 and GTCFR2 are not initialized before the GTMDR $n$ . Only GTCFR $n$ [RST $n$ ] and GTCFR $n$ [STP $n$ ] can be modified at any time.

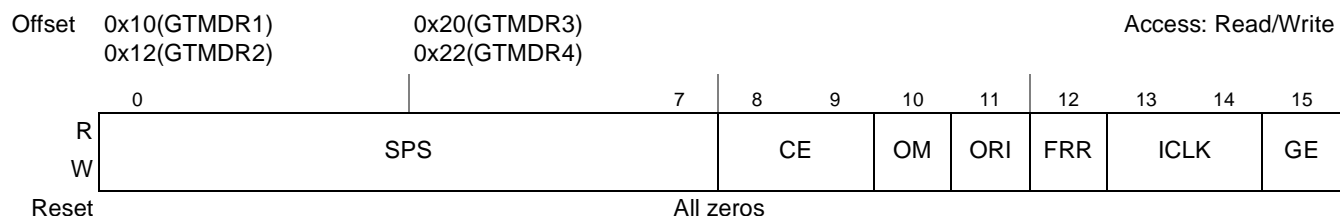


Figure 5-45. Global Timers Mode Registers (GTMDR1–GTMDR4)

Table 5-58 defines the bit fields of GTMDR.

Table 5-58. GTMDR Bit Settings

Bits	Name	Description
0–7	SPS	Secondary prescaler value The secondary prescaler is programmed to divide the clock input to corresponding timer by values from 1 to 256. The value 0x00 divides the clock by 1 and 0xFF divides the clock by 256.
8–9	CE	Capture edge and enable interrupt 00 Disable interrupt on capture event; capture function is disabled 01 Capture on rising TIN $n$ edge only and enable interrupt on capture event. 10 Capture on falling TIN $n$ edge only and enable interrupt on capture event. 11 Capture on any TIN $n$ edge and enable interrupt on capture event. <b>Note:</b> The frequency of TIN $n$ should be slower than system clock (TIN $n$ is sampled internally by system clock to detect TIN $n$ 's rising/falling edge before updating the counter)
10	OM	Output mode 0 Toggle $\overline{\text{TOUT}}_n$ every time when the corresponding timer matches its reference value. 1 Active-low pulse on $\overline{\text{TOUT}}_n$ for one timer input clock cycle (4 input clock cycles for the system clock) as defined by the ICLK $n$ bits. Thus, $\overline{\text{TOUT}}_n$ may be low for four general system clocks, one general system slow go clock period, or one TIN $n$ pin clock cycle period. <b>Note:</b> $\overline{\text{TOUT}}_n$ changes are internally synchronized to the rising edge of the system clock
11	ORI	Output reference interrupt enable 0 Disable interrupt for reference reached (does not affect interrupt on capture function). 1 Enable interrupt on reaching the reference value.
12	FRR	Free run/restart mode 0 Free run. The timer count continues to increment after the reference value is reached. 1 Restart. The timer count is reset immediately after the reference value is reached.

Table 5-58. GTMDR Bit Settings (continued)

Bits	Name	Description
13–14	ICLK	Input clock source for the timer. 00 Internally cascaded input. This selection means: For ICLK1, the timer 1 input is the output of timer 2. For ICLK2, the timer 1 input is the output of timer 2, the timer 2 input is the output of timer 3, the timer 3 input is the output of timer 4. For ICLK3, the timer 3 input is the output of timer 4. For ICLK4 this selection means no input clock is provided to the timer. 01 Internal general system bus clock. 10 Internal slow go clock (divided by 16 system bus clock). 11 TIN $n$ : corresponding TIN1, TIN2, TIN3, or TIN4 pin (falling edge).
15	GE	Gate enable 0 The $\overline{\text{TGATE}}_n$ signal is ignored. 1 The $\overline{\text{TGATE}}_n$ signal is used to control the timer.

### 5.6.5.3 Global Timers Reference Registers (GTRFR1–GTRFR4)

Global timers reference registers, shown in Figure 5-46, are 16-bit memory-mapped, read/write registers containing the 16-bit reference values for each timer's timeout. The reference value is not reached until  $\text{GTCNR}_n[\text{CNV}]$  increments to the value in  $\text{GTRFR}_n[\text{TRV}]$ .

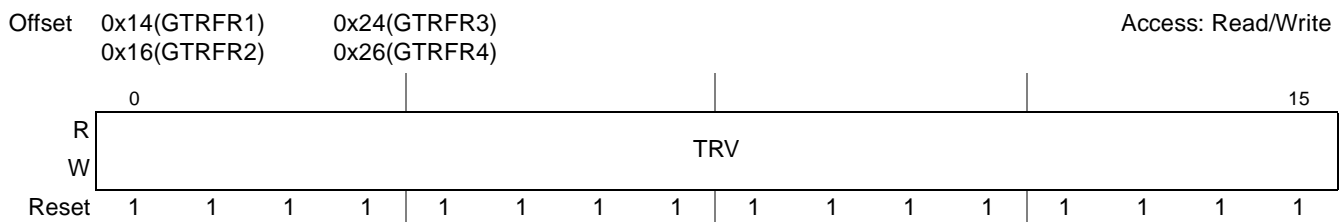


Figure 5-46. Global Timers Reference Registers (GTRFR1–GTRFR4)

Table 5-59 defines the bit fields of GTRFR.

Table 5-59. GTRFR Bit Settings

Bits	Name	Description
0–15	TRV	Timeout reference value. 16-bit timeout reference value for the corresponding timer. Set to all ones by reset.

### 5.6.5.4 Global Timers Capture Registers (GTCPR1–GTCPR4)

Global timers capture registers ( $\text{GTCPR}_1$ ,  $\text{GTCPR}_2$ ,  $\text{GTCPR}_3$ , and  $\text{GTCPR}_4$ ), shown in Figure 5-47, are used to latch the value of the counters according to  $\text{GTMDR}_n[\text{CE}]$ .

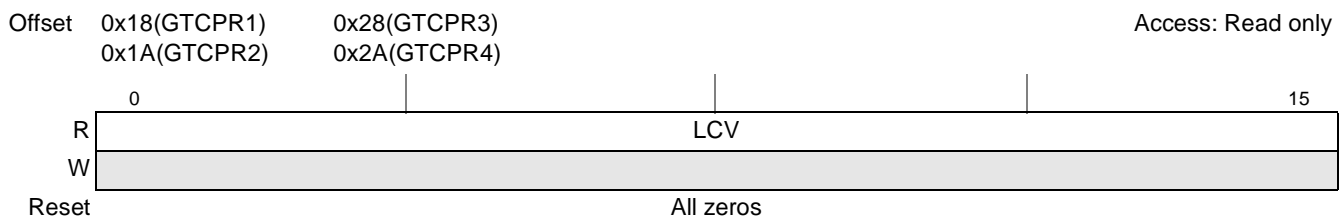


Figure 5-47. Global Timers Capture Registers (GTCPR1–GTCPR4)

Table 5-60 defines the bit fields of GTCPR $n$ .

Table 5-60. GTCPR $n$  Bit Settings

Bits	Name	Description
0–15	LCV	Latched counter value. Corresponding timer's 16-bit latched value.

### 5.6.5.5 Global Timers Counter Registers (GTCNR1–GTCNR4)

Global timers counter registers (GTCNR1, GTCNR2, GTCNR3, and GTCNR4), shown in Figure 5-48, are four 16-bit, memory-mapped, read/write up-counters. A read cycle to a GTCNR $n$ [CNV] fields yields the current value of the appropriate timer but does not affect the counting operation. A write cycle to a GTCNR $n$ [CNV] field sets the register to the written value, causing its corresponding primary and secondary prescaler counters to be reset.

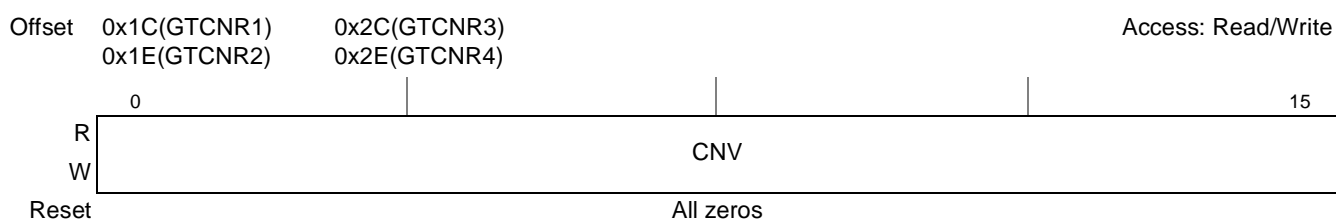


Figure 5-48. Global Timers Counter Registers (GTCNR1–GTCNR4)

Table 5-61 defines the bit fields of GTCNR.

Table 5-61. GTCNR Bit Settings

Bits	Name	Description
0–15	CNV	Counter value. Corresponding timer's 16-bit read/write up-counter value.

### 5.6.5.6 Global Timers Event Registers (GTEVR1–GTEVR4)

Global timers event registers (GTEVR1, GTEVR2, GTEVR3, and GTEVR4), shown in Figure 5-49, are used to report events recognized by any of the timers. On recognition of an output reference event, the appropriate timer sets GTEVR $n$ [REF], regardless of the corresponding GTMDR $n$ [ORI]. The capture event is only set if it is enabled by GTMDR $n$ [CE]. GTEVRs appear as memory-mapped registers to users, which can be read at any time.

GTEVR $n$  bits are cleared by writing ones to them (writing zeros does not affect bit values). Both bits must be reset before the timer negates the interrupt to the interrupt controller.

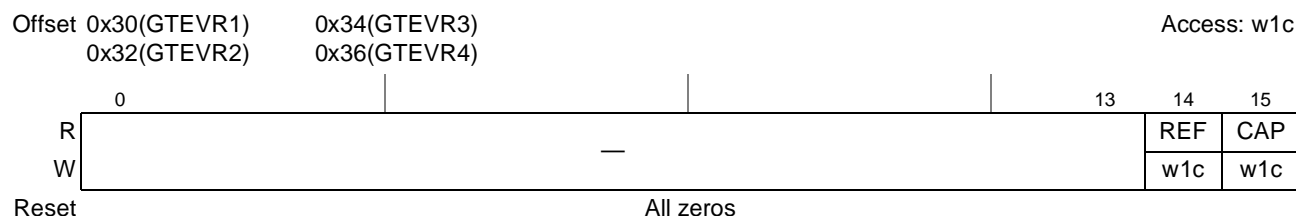


Figure 5-49. Global Timers Event Registers (GTEVR1–GTEVR4)

Table 5-62 defines the bit fields of  $GTEVR_n$ .

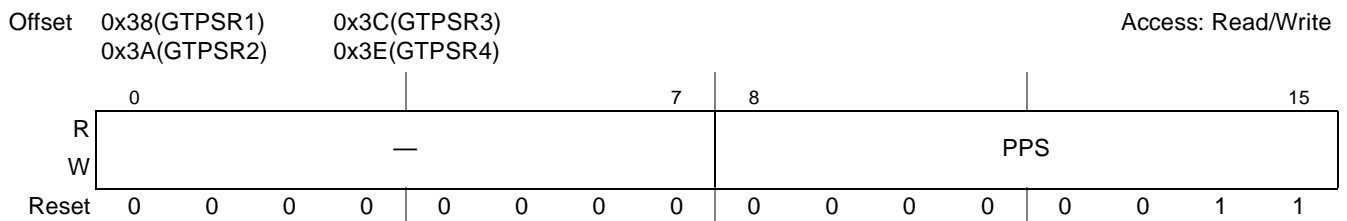
**Table 5-62.  $GTEVR_n$  Bit Settings**

Bits	Name	Description
0–13	—	Reserved, should be cleared.
14	REF	Output reference event 0 No event 1 The counter reached the $GTRFR_n[TRV]$ value. $GTMDR_n[ORI]$ is used to enable the interrupt request caused by this event.
15	CAP	Counter capture event Corresponding timer's 16-bit read/write up-counter value. 0 No event 1 The counter value has been latched into the $GTCPR_n[LCV]$ . $GTMDR_n[CE]$ is used to enable generation of this event.

### 5.6.5.7 Global Timers Prescale Registers (GTPSR1–GTPSR4)

The global timers prescale registers ( $GTPSR_1$ ,  $GTPSR_2$ ,  $GTPSR_3$ , and  $GTPSR_4$ ) are shown in Figure 5-50.

Erratic behavior may occur if  $GTPSR_n$  is not initialized before the corresponding  $GTMDR_n$ .



**Figure 5-50. Global Timers Prescale Registers ( $GTPSR_1$ – $GTPSR_4$ )**

Table 5-63 defines the bit fields of  $GTPSR_n$ .

**Table 5-63.  $GTPSR_n$  Bit Settings**

Bits	Name	Description
0–7	—	Reserved, should be cleared.
8–15	PPS	Primary prescaler bits The primary prescaler is programmed to divide the clock input to corresponding timer by values from 1 to 256. The value 0x00 divides the clock by 1 and 0xFF divides the clock by 256.

#### NOTE

The total timer prescale value is calculated as follows:

$$GTM_{n\_prescaler} = (GTPSR_n[PPS] + 1) \cdot (GTMDR_n[SPS] + 1)$$

This gives a total prescale range from 1 ( $GTPSR_n[PPS] = 0x00$ ,  $GTMDR_n[SPS] = 0x00$ ) to 65,536 ( $GTPSR_n[PPS] = 0xFF$ ,  $GTMDR_n[SPS] = 0xFF$ ).

## 5.6.6 Functional Description

This section provides a functional description of the general-purpose timer units, including detailed description of its modes of operation.

### 5.6.6.1 General-Purpose Timer Units

The clock input to the timer's prescaler can be selected from the following sources:

- The system clock
- The system slow go clock (internally divided by 16)

The general system clock is generated in the clock synthesizer and defaults to the system frequency. However, the general system clock has the option to be divided before it leaves the clock synthesizer. This mode, called slow go, is used to save power. Whatever the resulting frequency of the general system clock, the user can either choose that frequency or the frequency divided by 16 as the input to the prescaler of each timer. Alternatively, the user may prefer  $TIN_n$  to be the clock source.  $TIN_n$  is internally synchronized to the internal clock. If the user has chosen to internally cascade two 16-bit timers to a 32-bit timer, then a timer can use the clock generated by the output of another timer.

The clock input source is selected by the corresponding  $GTMDR_n[ICLK]$  bits. The prescalers ( $GTMDR_n[SPS]$  and  $GTPSR_n[PPS]$ ) can be programmed to divide the clock input by values from 1 to 65,536, and the output of the prescaler is used as an input to the 16-bit counters. The best resolution of the timer is one clock cycle (8 ns at a 125-MHz system clock, for example). The maximum period (when the reference value is all ones, the prescaler divides by  $256 \times 256$ , and slow go mode divides by 16) for one 16-bit timer is ~549.6 sec at 125 MHz.

### 5.6.6.2 Reference Modes

Each timer can be configured to count until a reference is reached and then either begin a new time count immediately or continue to run. The FRR bit of the corresponding  $GTMR$  selects each mode.

- Free run reference mode ( $GTMDR_n[FRR] = 0$ )  
The corresponding timer count continues to increment after the reference value is reached.
- Reset reference mode ( $GTMDR_n[FRR] = 1$ )  
The corresponding timer count is reset immediately after the reference value is reached.

Upon reaching the reference value, the corresponding  $GTEVR_n[REF]$  bit is set and an interrupt is issued if  $GTMDR_n[ORI] = 1$ . The timers can output a signal on the timer output pin  $\overline{TOUT}_n$  if the reference value is reached (selected by the corresponding  $GTMDR_n[OM]$ ). This signal can be an active-low pulse or a toggle of the current output. The output can also be connected internally to the input of another timer, resulting in a 32- or 64-bit timer.

### 5.6.6.3 Capture Modes

In addition, each timer has a 16-bit field in  $GTCPR$ , used to latch the value of the counter when a defined transition of  $TIN_n$  is sensed by the corresponding input capture edge detector. The timers may be gated/restarted by an external gate signals ( $TGATE_n$ ) that controls the timers. The type of transition

triggering the capture is selected by the corresponding  $GTMDR_n[CE]$  bits. Upon a capture or reference event, corresponding  $GTEVR_n[REF]$  or  $GTEVR_n[CAP]$  is set and a maskable interrupt request is issued to the interrupt controller.

- Normal gate mode enables the count on a falling edge of  $\overline{TGATE}$  and disables the count on the rising edge of  $\overline{TGATE}$ . This mode allows the timer to count conditionally, based on the state of  $\overline{TGATE}$ .
- The restart gate mode performs the same function as normal mode, except it also resets the counter on the falling edge of  $\overline{TGATE}$ .

This mode has applications in pulse interval measurement and bus monitoring as follows:

- Pulse measurement—The restart gate mode can measure a low pulse on  $\overline{TGATE}$ . The rising edge of  $\overline{TGATE}$  completes the measurement and if  $\overline{TGATE}_n$  is connected externally to  $TIN_n$ , it causes the timer to capture the count value and generate a rising-edge interrupt.
- Bus monitoring—The restart gate mode can detect a signal that is stuck abnormally low. The bus signal should be connected to  $\overline{TGATE}$ . The timer count is reset on the falling edge of the bus signal and if the bus signal does not go high again within the number of user-defined clocks, an interrupt can be generated.

The gate function is enabled in the  $GTMDR$ ; the gate operating mode is selected in the  $GTCFR_n$ .

#### NOTE

$\overline{TGATE}$  is internally synchronized to the system clock. If  $\overline{TGATE}$  meets the asynchronous input setup time, the counter begins or stops counting after one system clock when working with the internal clock.

#### 5.6.6.4 Cascaded Modes

$GTCFR_n[PCAS]$  and  $GTCFR2[SCAS]$  are used to put the timers into different cascaded modes:

- Non-cascaded mode ( $GTCFR_n[PCAS] = 0$  and  $GTCFR2[SCAS] = 0$ )

If  $GTCFR_n[PCAS] = 0$  and  $GTCFR2[SCAS] = 0$ , each timer (timer 1, timer 2, timer 3, and timer 4), function as a independent 16-bit timer with a 16-bit  $GTRFR$ ,  $GTCPR$ ,  $GTMDR$ , and  $GTCNR$  for each one (Figure 5-51). When working in the none-cascaded mode, the non-cascaded  $GTRFR$ ,  $GTCPR$ , and  $GTCNR$  should be referenced with appropriate 16-bit bus cycles.

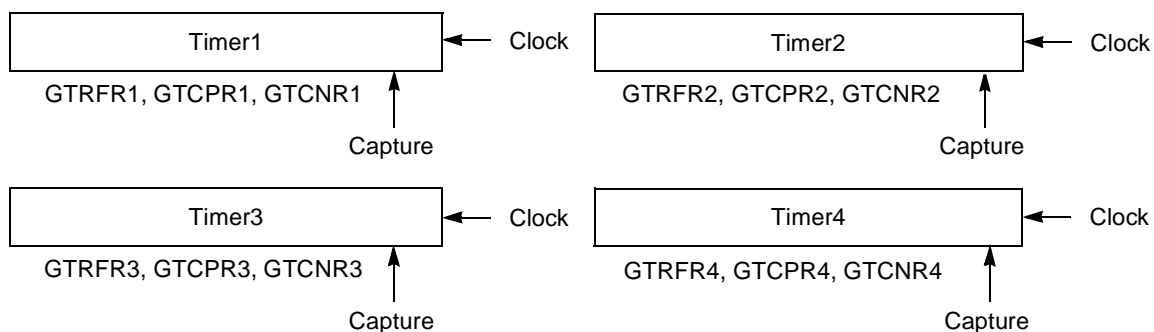


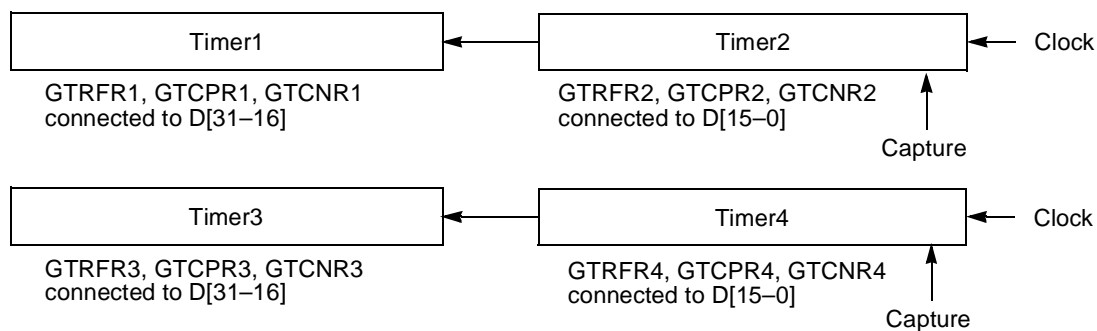
Figure 5-51. Timers Non-Cascaded Mode Block Diagram

- Pair-cascaded mode ( $GTCFR1[PCAS] = 1$  and/or  $GTCFR2[PCAS] = 1$ ,  $GTCFR2[SCAS] = 0$ )



In this mode, two 16-bit timers can be internally cascaded to form a 32-bit counter: timer 1 may be internally cascaded to timer 2 and timer 3 may be internally cascaded to timer 4, as shown in [Figure 5-52](#). Since the decision to cascade timers is made independently, the user has the option of selecting two 16-bit timers and one 32-bit timer ( $GTCFR1[PCAS] = 1$ ,  $GTCFR2[PCAS] = 0$  or  $GTCFR1[PCAS1] = 0$ ,  $GTCFR2[PCAS] = 1$ ), or two 32-bit timers ( $GTCFR1[PCAS] = 1$  and  $GTCFR2[PCAS] = 1$ ).

If  $GTCFR1[PCAS] = 1$  and/or  $GTCFR2[SCAS] = 1$ , the two 16-bit timers (timer 1 and timer 2 or timer 3 and timer 4) function as a 32-bit timer with a 32-bit  $GTRFR$ ,  $GTCPR$ , and  $GTCNR$ . In this case,  $GTMDR1/GTMDR3$  is ignored, and the modes and functions are defined using  $GTMDR2/GTMDR4$  and  $GTCFR1/GTCFR2$ . The capture are controlled from  $TIN2$ , and the interrupts are generated from  $GTEVR2$ . When working in the pair-cascaded mode, the cascaded  $GTRFR$ ,  $GTCPR$ , and  $GTCNR$  should be referenced with 32-bit bus cycles.

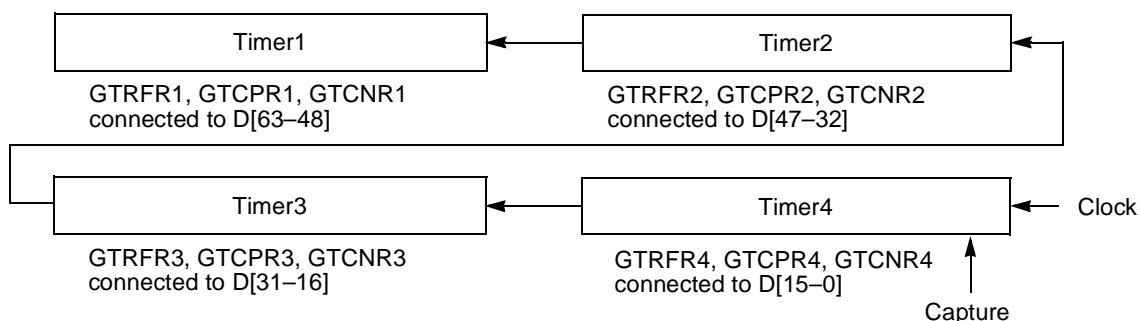


**Figure 5-52. Timer Pair-Cascaded Mode Block Diagram**

- Super-cascaded mode ( $GTCFR2[SCAS] = 1$ )

In this mode, all four 16-bit timers can be internally cascaded to form a 64-bit counter, as shown in [Figure 5-53](#).

If  $GTCFR2[SCAS] = 1$ , the all four 16-bit timers function as a 64-bit timer with a cascaded 32-bit  $GTRFR$ ,  $GTCPR$ , and  $GTCNR$ . In this case, registers  $GTMDR1$ ,  $GTMDR2$ ,  $GTMDR3$ , and  $GTCFR1$  are ignored, and the modes and functions are defined using  $GTMDR4$  and  $GTCFR2$  only. The capture are controlled from  $TIN4$ , and the interrupts are generated from  $GTEVR4$ . When working in the super-cascaded mode, the cascaded  $GTRFR$ ,  $GTCPR$ , and  $GTCNR$  should be referenced with two 32-bit bus cycles.



**Figure 5-53. Timers Super-Cascaded Mode Block Diagram**

## 5.6.7 Initialization/Application Information (Programming Guidelines for GTM Registers)

The following initialization sequence of GTM is recommended:

- Write to  $GTCFR_n$  in order to reset, to stop, or to configure the appropriate timer's operation: cascaded timers configuration, gate mode configuration.
- Write to  $GTPSR_n[PPS]$  fields in order to program the appropriate timer's clock primary prescaler.
- Write to  $GTMDR_n$  in order to choose an input clock, to program the secondary prescaler, and to set a desirable appropriate timer's operational mode.

### NOTE

Erratic behavior may occur if  $GTCFR_n$  and  $GTPSR$  are not initialized before the  $GTMDR$ . Only  $GTCFR_n[RST_n]$  can be modified at any time

- Clear  $GTEVR_n[REF]$  and  $GTEVR_n[CAP]$  by writing 1s in order to clear the previous events.
- Write to  $GTRFR$  and to  $GTCNR_n$  according to appropriate timer's  $GTMDR_n$  programming.

### NOTE

A write cycle to a  $GTCNR_n[CNV]$  fields sets the register to the written value, causing its corresponding primary and secondary prescalers, ( $GTPSR_n[PPS]$  and  $GTMDR_n[SPS]$ ), to be reset.

- Write to  $GTCFR_n[STP_n]$  and to  $GTCFR_n[RST_n]$  in order to initialize the appropriate timer's operation.
- Write to  $GTCFR_n$  register in order to start the corresponding timer ( $GTCFR_n[STP_n] = 0$ ).

## 5.7 Power Management Control (PMC)

The device provides a power management control (PMC) unit, which enables the device to smoothly enter and exit low-power modes. Low-power modes may be used when internal units in the device temporarily or permanently do not perform any action.

The device uses one or more of the following methods for power saving:

- DDR2 power management
- Shutting down clocks to unused blocks
- Software-controlled power-down states for the e300 core

## 5.7.1 External Signal Description

Table 5-64 describes the power management signals.

**Table 5-64. System Control Signals—Detailed Signal Descriptions**

Signal	I/O	Description				
$\overline{\text{QUIESCE}}$	O	Quiesce state. Indicates that the processor system and e300 core are in low- power state.				
		<table border="1"> <tr> <td><b>State Meaning</b></td> <td>Asserted—The system and e300 core are in low-power state. Negated—The system and e300 core are not in low-power state.</td> </tr> <tr> <td><b>Timing</b></td> <td>The timing between a quiesce request from the e300 core and the assertion of the external indication or between negation of the e300 core's quiesce request and negation of the external indication depends on the current state of the internal system units and may vary accordingly.</td> </tr> </table>	<b>State Meaning</b>	Asserted—The system and e300 core are in low-power state. Negated—The system and e300 core are not in low-power state.	<b>Timing</b>	The timing between a quiesce request from the e300 core and the assertion of the external indication or between negation of the e300 core's quiesce request and negation of the external indication depends on the current state of the internal system units and may vary accordingly.
<b>State Meaning</b>	Asserted—The system and e300 core are in low-power state. Negated—The system and e300 core are not in low-power state.					
<b>Timing</b>	The timing between a quiesce request from the e300 core and the assertion of the external indication or between negation of the e300 core's quiesce request and negation of the external indication depends on the current state of the internal system units and may vary accordingly.					

## 5.7.2 PMC Memory Map/Register Definition

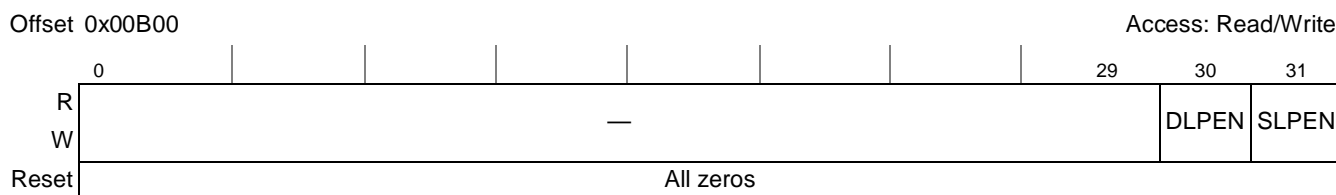
Table 5-65 shows the memory map for the power management controller registers.

**Table 5-65. Power Management Controller Registers Memory Map**

Offset	Register	Access	Reset	Section/Page
<b>PMC—Block Base Address 0x0_0B00</b>				
0x00B00	Power management controller configuration register (PMCCR)	R/W	0x0000_0000	<a href="#">5.7.2.1/5-69</a>
0x00B04– 0x00BFC	Reserved	—	—	—

### 5.7.2.1 Power Management Controller Configuration Register (PMCCR)

The power management controller configuration register (PMCCR), shown in Figure 5-54, controls whether only the e300 core enters low-power state upon quiesce request or additional parts of the device also enters low-power state.



**Figure 5-54. Power Management Controller Configuration Register**

Table 5-66 defines the bit fields of PMCCR.

**Table 5-66. PMCCR Bit Settings**

Bits	Name	Description
0–29	—	Reserved. Write has no effect, read returns 0.
30	DLPEN	DDR SDRAM low power enable 0 The DDR SDRAM memory controller is prevented from entering low-power state. 1 The DDR SDRAM memory controller enters low-power state when the rest of the system enters low-power, according to SLPEN setting. DDR SDRAM enters self-refresh mode (if enabled by DDR_SDRAM_CFG[SREN] memory controller register) and DDR clocks (MCK <sub>n</sub> ) are shut off. This bit is cleared when the device exits from low-power state. Note that setting this bit without setting SLPEN has no effect.
31	SLPEN	System low power enable 0 The system is prevented from entering low-power state. 1 The system enters low-power state when a quiesce signal is generated from e300 core. This bit is cleared when the device exits from low-power state.

### 5.7.3 Functional Description

The device has features to minimize power consumption at several levels. Software can shut down clocks to individual blocks when they are not needed through a memory-mapped register in the clock unit (SCCR). Additionally, software running on the e300 core can access the e300 core's SPRs to put the e300 core into doze, nap, or sleep power-down states. These power management features are described in further detail in this section.

There are four power states in MPC8308:

- D0: Full-power state
- D1: Core in doze mode, e300 PLL running
- D2: Core in nap mode, e300 PLL running
- D3: Core in sleep mode, e300 PLL not running

**Table 5-67. Software-Controller Power-Down States—Basic Description**

System Mode	Core Mode	Description	Core Responds To		DDR SDRAM State	Quiesce Signal State
			Snoop	Interrupt		
Full On (PMCCR[SLPEN] = 0)	Full On	The e300 core is operating normally	Yes	Yes	Active	Negated
	Doze	Core stops dispatching new instructions (core is halted), and most of the core functional units are disabled. System operates normally.	Yes	Yes	Active	Negated
	Nap	Core is stopped with its clocks off except to time base. System operates normally.	No	Yes	Active	Negated
	Sleep	Core is stopped with its clocks off. Core clocks to all blocks (including core time base) are stopped except to the interrupt unit. System operates normally.				

Table 5-67. Software-Controller Power-Down States—Basic Description (continued)

System Mode	Core Mode	Description	Core Responds To		DDR SDRAM State	Quiesce Signal State
			Snoop	Interrupt		
Low Power (PMCCR[SLPEN] = 1)	Nap	Core operation as described above. System is in idle state, DDR SDRAM memory operates in self-refresh mode if enabled.	No	Yes	According to PMCCR [DLPEN]	Asserted
	Sleep					

### 5.7.3.1 Shutting Down Clocks to Unused Blocks

As described in [Section 4.5.2.3, “System Clock Control Register \(SCCR\),”](#) SCCR provides a way to shut down certain functional blocks within the device when they are not needed in a particular system. SCCR can be written by the e300 core or by an external master. Powering down a block in this way turns off all clocks to that block. It does not remove power. It is required that the SCCR is written to shut down a certain functional block only when that block is idle.

#### NOTE

Functional blocks disabled using SCCR cannot respond to configuration accesses. Any access to configuration, control, and status registers of a disabled block is a programming error.

### 5.7.3.2 Software-Controlled Power-Down States

e300 core software can place the core in doze, nap, or sleep power-down states by writing to HID0 in the core, as described in detail in the section “Hardware Implementation Register 0 (HID0),” of the *e300 PowerPC Core Reference Manual*. In addition, if PMCCR[SLPEN] is set when the e300 core request to enter nap or sleep modes, it also causes the DDR2 controller to enter low-power mode. The basic relationships between core and system power management states is shown in [Table 5-67](#).

To preserve cache coherency and otherwise avoid loss of system state, the core transitions to low-power modes is coordinated with DDR2 controller. The power management controller allows the core to enter power-down mode only when the rest of the system is idle.

When the power management controller detects that the internal system bus is idle, and there are no outstanding transactions, it signals the internal logic units to enter low-power state.

If PMCCR[DLPEN] and/or PMCCR[SLPEN] is set, the DDR SDRAM is first set to self-refresh mode (if enabled by DDR\_SDRAM\_CFG[SREN] memory controller register) before the memory controller stops driving refresh commands. Self-refresh mode guarantees that the memory content remains valid while the memory controller and its clocks are off. The DDR clocks are then disabled. The DLL is then shut off and stops driving clocks on MCKn pins. Finally the DDR SDRAM memory controller enters low-power state and acknowledges the power management controller.

The power management controller then signals the core and acknowledges its request to enter power-down mode. Finally the  $\overline{\text{QUIESCE}}$  output signal is asserted.

### 5.7.3.3 Exiting Core and System Low-Power States

The device can exit low-power state and return to full-on mode for one of the following reasons:

- The core internal time base unit invokes a request to exit low-power state.
- The power management controller has detected that the system is not idle and there are outstanding transactions on the internal bus.

The actions taken to exit low-power state depend on the mode and whether the system or only the core are in this state. The following sections describe the various scenarios.

#### 5.7.3.3.1 Exiting Low-Power States—Core-Only Mode

Exit from doze mode is controlled only by the core itself and does not involve the power management controller or other blocks in the device.

Nap mode is exited according to the internal time base unit of the core. When the core returns to full-on state, it signals to the power management controller that it is ready and is immediately acknowledged to access the DDR2 controller.

#### 5.7.3.3.2 Exiting Low-Power States—Core and System Mode

The power management controller decides to exit low-power state when it detects that the system is not idle anymore. The device may exit idle state when one of the peripheral interfaces makes a request to access the internal bus or when the core returns to full-on state, as described above, and makes a request to access the internal bus. For example, eTSEC wants to read from or write into DDR2 SDRAM memory.

If the particular DDR SDRAM memory controller is in low-power state (PMCCR[DLPEN] was set when entering low-power state), the power management controller initially enables the DDR SDRAM memory controller and its DLL, allowing it to lock. When locked, DDR SDRAM clocks (MCK $n$ ) are enabled and the memory controller exits self-refresh and returns to auto-refresh mode.

The power management controller then enables other internal units and interrupts the e300 core. When all internal units, including the e300 core, are ready, the power management controller enables the device to return to full-on state, negates the  $\overline{\text{QUIESCE}}$  output, and clears PMCCR[SLPEN]. Outstanding requests for transactions are now granted to execute on the internal bus.

# Chapter 6

## Arbiter and Bus Monitor

This chapter describes operation theory of the arbiter in the device. In addition, it describes configuration, control, and status registers of the arbiter.

### 6.1 Overview

The arbiter is responsible for providing coherent system bus arbitration. It tracks all address and data tenures and provides all the arbitration signals to masters and slaves. In addition, it monitors the bus and reports on errors and protocol violations.

The arbiter includes the following features:

- Supports a programmable pipeline depth (from 1 to 4)
- Supports four levels of priority for bus arbitration
- Supports repeat-request mode: number of programmable consecutive transactions from the same master (up to eight transactions)
- Supports data streaming operations
- Supports programmable address bus parking mode: disable, park to last bus owner, park to software selected master
- Claims address only, reserved and illegal transaction types, report on it and can raise maskable interrupt
- Provides timers for address tenure time out and data tenure time out detection and can issue maskable interrupt, if any timer expired
- Reports on transfer error and can issue maskable interrupt
- Can issue regular or machine check interrupt for each type of error event (programmable)

#### 6.1.1 Coherent System Bus Overview

The coherent system bus is the central bus of the device. Any data transaction from master to slave in the device passes through the coherent system bus. The device coherent system bus supports pipelined transactions. It has independent address and data tenures. Pipeline depth determines the number of address tenures that can be started before the first data tenure is finished.

Basic burst size is equal to cache line length of the core, which is 32 bytes. Using repeat request mode enables up to eight consecutive bursts to be executed by the same master. The maximum number of

consecutive transactions can be limited by programming arbiter configuration register. See [Section 6.2.1, “Arbiter Configuration Register \(ACR\),”](#) for more details.

### NOTE

Write accesses to different interfaces are not guaranteed to finish in order.

## 6.2 Arbiter Memory Map/Register Definition

[Table 6-1](#) shows the memory map for arbiter’s configuration, control, and status registers.

**Table 6-1. Arbiter Register Map**

System Arbiter—Block Base Address 0x0_0800				
Memory Offset (Hex)	Register	Access	Reset	Section/Page
0x00	Arbiter configuration register (ACR)	R/W	0x0000_0000/ 0x0010_0000 <sup>1</sup>	<a href="#">6.2.1/6-3</a>
0x04	Arbiter timers register (ATR)	R/W	FFFF_FFFF	<a href="#">6.2.2/6-4</a>
0x0C	Arbiter event register (AER)	w1c	0x0000_0000	<a href="#">6.2.3/6-5</a>
0x10	Arbiter interrupt definition register (AIDR)	R/W	0x0000_0000	<a href="#">6.2.4/6-6</a>
0x14	Arbiter mask register (AMR)	R/W	0x0000_0000	<a href="#">6.2.5/6-7</a>
0x18	Arbiter event attributes register (AEATR)	R	0x0000_0000 <sup>2</sup>	<a href="#">6.2.6/6-8</a>
0x1C	Arbiter event address register (AEADR)	R	0x0000_0000 <sup>2</sup>	<a href="#">6.2.7/6-9</a>
0x20	Arbiter event response register (AERR)	R/W	0x0000_0000	<a href="#">6.2.8/6-10</a>

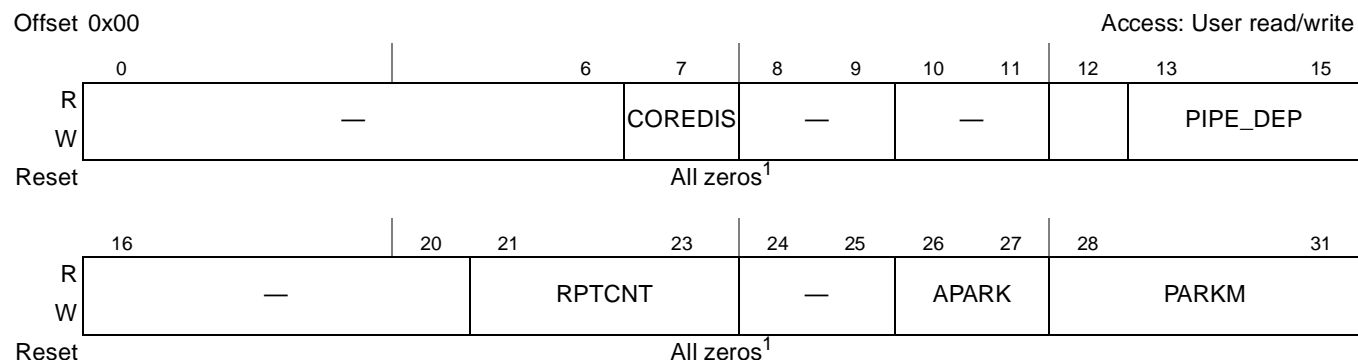
<sup>1</sup> Reset value is determined from the core PLL configuration of the reset configuration word. See [Chapter 4, “Reset, Clocking, and Initialization,”](#) for details.

<sup>2</sup> The registers AEATR and AEADR are affected only by the assertion of  $\overline{\text{PORESET}}$ .



## 6.2.1 Arbiter Configuration Register (ACR)

The arbiter configuration register (ACR) defines the arbiter modes and parked master on the bus. Figure 6-1 shows the fields of ACR.



<sup>1</sup> Note that the reset value of COREDIS and bits 10–11 are determined from reset configuration word. (See Section 4.3.2, “Reset Configuration Words,” for more details on reset configuration word.)

**Figure 6-1. Arbiter Configuration Register (ACR)**

Table 6-2 describes ACR fields.

**Table 6-2. ACR Field Descriptions**

Bits	Name	Description
0–6	—	Write reserved, read = 0
7	COREDIS	Core disable. Specifies whether CPU is disabled. When CPU is disabled, it cannot be granted on the bus by the arbiter. After reset, this bit receives its value from the reset configuration bit of COREDIS and can be configured by software. Also, if boot source is boot sequencer, COREDIS must be set to 1 at reset and the last transaction of the boot sequencer must set COREDIS to 0, if CPU enable is needed. 0 CPU enabled. 1 CPU disabled.
8–9	—	Write reserved, read = 0
10–11	—	Reserved. Write should preserve reset value. The reset value is a function of the core PLL configuration, which is part of the reset configuration word. When the core is set to operate at 1:1 or 3:2 bus clock, these bits are set to ‘01’ during reset; otherwise, they are set to ‘00’.
12	—	Write reserved, read = 0
13–15	PIPE_DEP	Pipeline depth (number of outstanding transactions). 000 Pipeline depth 1 (1 outstanding transaction) 001 Pipeline depth 2 (2 outstanding transactions) 010 Pipeline depth 3 (3 outstanding transactions) 011 Pipeline depth 4 (4 outstanding transactions) 1xx Reserved
16–20	—	Write reserved, read = 0

Table 6-2. ACR Field Descriptions (continued)

Bits	Name	Description
21–23	RPTCNT	Repeat count. Specifies the maximum number of consecutive transactions, that any master can perform, using $\overline{\text{REPEAT}}$ request mode. 000 1 consecutive transactions ( $\overline{\text{REPEAT}}$ request mode disable) 001 2 consecutive transactions 010 3 consecutive transactions 011 4 consecutive transactions 100 5 consecutive transactions 101 6 consecutive transactions 110 7 consecutive transactions 111 8 consecutive transactions <b>Note:</b> It is recommended not to program this field for more than four consecutive transactions.
24–25	—	Write reserved, read = 0
26–27	APARK	Address parking. Specifies arbiter bus parking mode. 00 Park to master. Arbiter parks the address bus to the master, that is selected by numeric value of PARKM field. 01 Park to last owner. Arbiter parks the address bus to last bus owner. 10 Disable. Arbiter does not assert $\overline{\text{BG}}$ to any master, if no $\overline{\text{BR}}$ is present. 11 Reserved
28–31	PARKM	Parking master. 0000 e300 core 0001 Reserved 0010 TSEC1, TSEC2 0011 Reserved 0100 DMA, eSDHC, USB 0101 PCI Express 0110–1111 Reserved

## 6.2.2 Arbiter Timers Register (ATR)

The arbiter timers register (ATR) defines the arbiter address time out (ATO) and data time out (DTO) values. Figure 6-2 shows the fields of ATR.

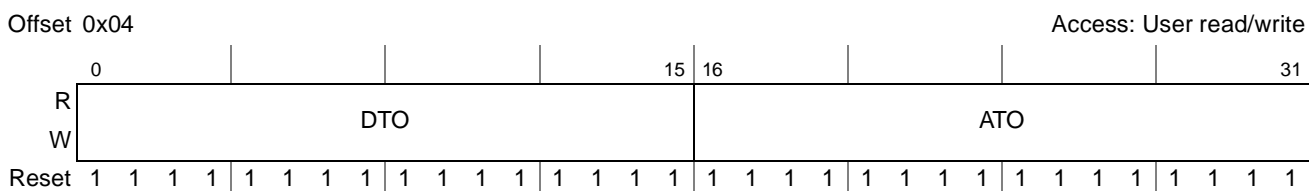


Figure 6-2. Arbiter Timers Register (ATR)

Table 6-3 describes ATR fields.

**Table 6-3. ATR Field Descriptions**

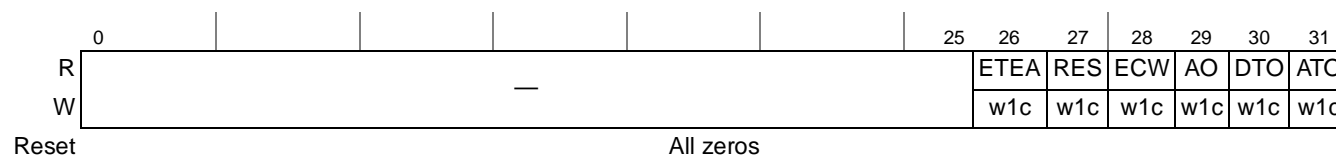
Bits	Name	Description
0–15	DTO	Data time out. Specifies the time-out period for the data tenure. The granularity of this field is 128 bus clocks. The maximum value is 8355840 coherent system bus clocks. Data time_out occurs if the data tenure does not end before the specified time-out period. When DTO = n, the timeout cycle is $n \times 128$ . 0000 Reserved 0001 128 clock cycles 0002 256 clock cycles 0003 384 clock cycles ... FFFF 8355840 clock cycles
16–31	ATO	Address time out. Specifies the time-out period for the address tenure. The granularity of this field is 128 bus clocks. Maximum value is 8355840 coherent system bus clocks. Address time-out occurs if the address tenure did not end before the specified time-out period. When ATO = n, the timeout cycle is $n \times 128$ . 0000 Reserved 0001 128 clock cycles 0002 256 clock cycles 0003 384 clock cycles ... FFFF 8355840 clock cycles

## 6.2.3 Arbiter Event Register (AER)

The arbiter uses arbiter event register (AER) to report on erroneous transactions. This register is cleared by writing ones to the fields to be cleared. Figure 6-3 shows the fields of AER.

Offset 0x0C

Access: User w1c



**Figure 6-3. Arbiter Event Register (AER)**

Table 6-4 describes AER fields.

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

Bits	Name	Description
0–25	—	Write reserved, read = 0
26	EAEA	Transfer error. Reports on detection of transfer error by one of the slaves. 0 No transfer error detected by one of the slaves. 1 Transfer error detected by one of the slaves.
27	RES	Reserved transfer type. Reports on transaction with reserved transfer type. See Section 6.3.2.5, “Reserved Transaction Type,” for more information. 0 No transaction with reserved transfer type occurred. 1 Transaction with reserved transfer type occurred.

Table 6-4. AER Field Descriptions (continued)

Bits	Name	Description
28	ECW	External control word transfer type. Reports on transaction with external control word transfer type. See <a href="#">Section 6.3.2.6, “Illegal (eciwx/ecowx) Transaction Type,”</a> for more information. 0 No transaction with external control word transfer type occurred. 1 Transaction with external control word transfer type occurred.
29	AO	Address Only transfer type. Reports on transaction with address only transfer type. See <a href="#">Section 6.3.2.4, “Address Only Transaction Type,”</a> for more information. 0 No transaction with address only transfer type occurred. 1 Transaction with address only transfer type occurred.
30	DTO	Data time out. Reports on data tenure time out. 0 Data time out timer is not expired. 1 Data time out timer is expired.
31	ATO	Address time out. Reports on address tenure time out. 0 Address time out timer is not expired. 1 Address time out timer is expired.

## 6.2.4 Arbiter Interrupt Definition Register (AIDR)

The arbiter interrupt definition register (AIDR) determines the interrupt that responds to different error conditions. Setting a bit defines the corresponding interrupt as MCP interrupt; clearing a bit defines the corresponding interrupt as regular interrupt. [Figure 6-4](#) shows the fields of AIDR.

Offset 0x10

Access: User read/write

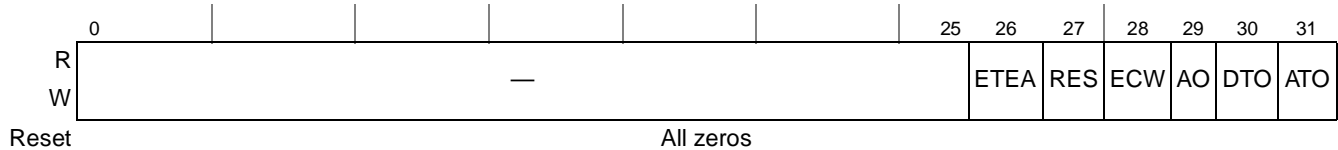


Figure 6-4. Arbiter Interrupt Definition Register (AIDR)

[Table 6-5](#) describes AIDR fields.

Table 6-5. AIDR Field Descriptions

Bits	Name	Description
0–25	—	Write reserved, read = 0
26	ETEA	Transfer error. Detection of transfer error by one of the slaves interrupt definition. 0 Detection of transfer error by one of the slaves causes regular interrupt. 1 Detection of transfer error by one of the slaves causes MCP interrupt.
27	RES	Reserved transfer type. Transaction with reserved transfer type interrupt definition. 0 Transaction with reserved transfer type causes regular interrupt. 1 Transaction with reserved transfer type causes MCP interrupt.
28	ECW	External control word transfer type. Transaction with external control word transfer type interrupt definition. 0 Transaction with external control word transfer type causes regular interrupt. 1 Transaction with external control word transfer type causes MCP interrupt.

Table 6-5. ADR Field Descriptions (continued)

Bits	Name	Description
29	AO	Address only transfer type. Transaction with address only transfer type interrupt definition. 0 Transaction with address only transfer type causes regular interrupt. 1 Transaction with address only transfer type causes MCP interrupt.
30	DTO	Data time out. Data tenure time out interrupt definition. 0 Data tenure time out causes regular interrupt. 1 Data tenure time out causes MCP interrupt.
31	ATO	Address time out. Address tenure time out interrupt definition. 0 Address tenure time out causes regular interrupt. 1 Address tenure time out causes MCP interrupt.

## 6.2.5 Arbiter Mask Register (AMR)

The arbiter mask register (AMR) is used to mask interrupts or reset requests. Setting a mask bit enables the corresponding interrupt or reset request; clearing a bit masks it. Regular interrupts, MCP interrupts, and reset requests can be masked by AMR register. Figure 6-5 shows the fields of AMR.

Offset 0x14

Access: User read/write



Figure 6-5. Arbiter Mask Register (AMR)

Table 6-6 describes AMR fields.

Table 6-6. AMR Field Descriptions

Bits	Name	Description
0–25	—	Write reserved, read = 0
26	ETEA	Transfer error. Detection of transfer error by one of the slaves interrupt mask bit. 0 Detection of transfer error by one of the slaves interrupt disabled. 1 Detection of transfer error by one of the slaves interrupt enabled.
27	RES	Reserved transfer type. Transaction with reserved transfer type interrupt mask bit. 0 Transaction with reserved transfer type interrupt disabled. 1 Transaction with reserved transfer type interrupt enabled.
28	ECW	External control word transfer type. Transaction with external control word transfer type interrupt mask bit. 0 Transaction with external control word transfer type interrupt disabled. 1 Transaction with external control word transfer type interrupt enabled.
29	AO	Address only transfer type. Transaction with address only transfer type interrupt mask bit. 0 Transaction with address only transfer type interrupt disabled. 1 Transaction with address only transfer type interrupt enabled.

Table 6-6. AMR Field Descriptions (continued)

Bits	Name	Description
30	DTO	Data time out. Data tenure time out interrupt mask bit. 0 Data tenure time out interrupt disabled. 1 Data tenure time out interrupt enabled.
31	ATO	Address time out. Address tenure time out interrupt mask bit. 0 Address tenure time out interrupt disabled. 1 Address tenure time out interrupt enabled.

## 6.2.6 Arbiter Event Attributes Register (AEATR)

The arbiter event attributes register (AEATR) reports the type of transaction that causes error that is specified in the event register. See [Section 6.2.3, “Arbiter Event Register \(AER\),”](#) for more information. AEATR is cleared only by power-on reset. The attributes of the first error event are stored. Note that this means that AEATR does not change its value when AER is not clear. As AEATR is not affected by soft or hard reset, software can read this register and determine the cause of the bus failure, even if the failure caused a deadlock situation. For more information, see [Section 6.4.2, “Error Handling Sequence.”](#)

Figure 6-6 shows the fields of AEATR.



Figure 6-6. Arbiter Event Attributes Register (AEATR)

Table 6-7 describes AEATR fields.

Table 6-7. AEATR Field Descriptions

Bits	Name	Description
0–4	—	Write reserved, read = 0
5–7	EVENT	Event type. 000 Address time out 001 Data time out 010 Address only transfer type 011 External control word transfer type 100 Reserved transfer type 101 Transfer error 11c Reserved
8–10	—	Write reserved, read = 0

Table 6-7. AEATR Field Descriptions (continued)

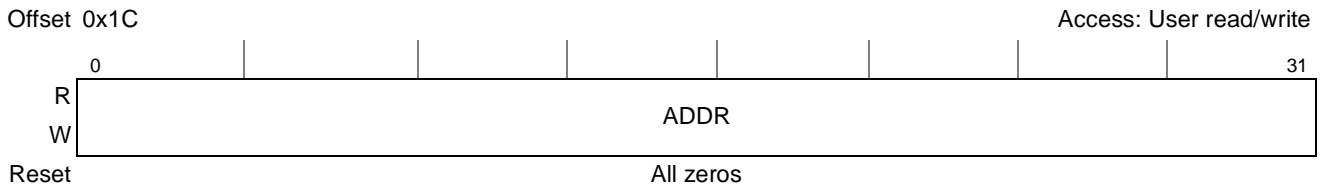
Bits	Name	Description
11–15	MSTR_ID	Master ID. 00000 e300 core data transaction                      01001 I2C (boot sequencer) 00001 Reserved    01010 JTAG 00010 e300 core instruction fetch                      01011 Reserved 00011 Reserved    01100 eSDHC 00100 TSEC1     01101–11100 Reserved 00101 TSEC2     11101 PCI Express 00110 Reserved    11110 Reserved 00111 USB     11111 DMA 01000 Reserved  <b>Note:</b> Master ID reflects the source of transaction and is used for debug purpose.
16–19	—	Write reserved, read = 0
20	TBST	Transfer burst. 0 Burst transaction. Transfer size is greater than 8 bytes 1 Single-beat transaction. Transfer size is up to 8 bytes
21–23	TSIZE	Transfer size. Transfer size encoding depends on the value of the field TBST. TBST = 1:    TBST = 0: 001 1 byte    000 16 bytes 010 2 bytes     001 24 bytes 011 3 bytes     010 32 bytes 100 4 bytes     011–111 Reserved 101 5 bytes 110 6 bytes 111 7 bytes 000 8 bytes
24–26	—	Write reserved, read = 0
27–31	TTYPE	Transfer type. 00000 Address-only    01111 Reserved 00001 Address-only    10000 Address-only 00010 Single-beat or burst write                              1XX01 Reserved 00011 Reserved    10010 Single-beat write 00100 Address-only    1XX11 Reserved 00101 Reserved    10100 <b>ecowx</b> —Illegal single-beat write 00110 Burst write     10110 Reserved 00111 Reserved    11000 Address-only 0100x Address-only    11010 Single-beat or burst read 0101x Single-beat or burst read                              11100 <b>eciwx</b> —Illegal single-beat read 0110x Address-only    11110 Burst read 01110 Burst read

## 6.2.7 Arbiter Event Address Register (AEADR)

The arbiter event address register (AEADR) reports the address of transaction that causes the error that is specified in the event register. See [Section 6.2.3, “Arbiter Event Register \(AER\),”](#) for more information. AEADR is cleared only by power-on reset. The address of the first error event is stored. Note that this means that AEADR does not change its value when AER is not clear. As AEADR is not affected by soft or hard reset, the software can read this register and determine the cause of the bus failure, even if the bus

failure had caused a deadlock situation. For more information, see [Section 6.4.2, “Error Handling Sequence.”](#)

Figure 6-7 shows the fields of AEADR.



**Figure 6-7. Arbiter Event Address Register (AEADR)**

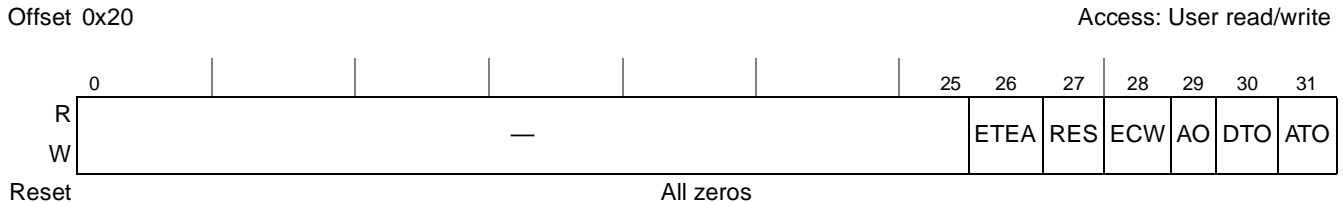
Table 6-8 describes AEADR fields.

**Table 6-8. AEADR Field Descriptions**

Bits	Name	Description
0–31	ADDR	Address of the event reported in AEATR register. See <a href="#">Section 6.2.6, “Arbiter Event Attributes Register (AEATR),”</a> for more information.

## 6.2.8 Arbiter Event Response Register (AERR)

The arbiter event response register (AERR) determines whether different error conditions cause interrupt or reset request. Setting a bit defines the corresponding error condition to cause reset request; clearing a bit defines the corresponding error condition to cause interrupt. [Figure 6-8](#) shows the fields of AERR.



**Figure 6-8. Arbiter Event Response Register (AERR)**

Table 6-9 describes AERR field.

**Table 6-9. AERR Field Descriptions**

Bits	Name	Description
0–25	—	Write reserved, read = 0
26	ETEA	Transfer error. Detection of transfer error by one of the slaves event response. 0 Detection of transfer error by one of the slaves causes interrupt. 1 Detection of transfer error by one of the slaves causes reset request.
27	RES	Reserved transfer type. Transaction with reserved transfer type interrupt definition. 0 Transaction with reserved transfer type causes interrupt. 1 Transaction with reserved transfer type causes reset request.



Table 6-9. AERR Field Descriptions

Bits	Name	Description
28	ECW	External control word transfer type. Transaction with external control word transfer type interrupt definition. 0 Transaction with external control word transfer type causes interrupt. 1 Transaction with external control word transfer type causes reset request.
29	AO	Address only transfer type. Transaction with address only transfer type interrupt definition. 0 Transaction with address only transfer type causes interrupt. 1 Transaction with address only transfer type causes reset request.
30	DTO	Data time out. Data tenure time out interrupt definition. 0 Data tenure time out causes interrupt. 1 Data tenure time out causes reset request.
31	ATO	Address time out. Address tenure time out interrupt definition. 0 Address tenure time out causes interrupt. 1 Address tenure time out causes reset request.

## 6.3 Functional Description

The following sections describe arbitration policy and bus error detection.

### 6.3.1 Arbitration Policy

The arbitration process involves the masters and the arbiter. Masters arbitrate on the privilege to own an address tenure. For data tenures, the arbiter uses the same order of transactions as address tenures.

Figure 6-9 shows the interface signals between the arbiter and masters that are involved in address bus arbitration.

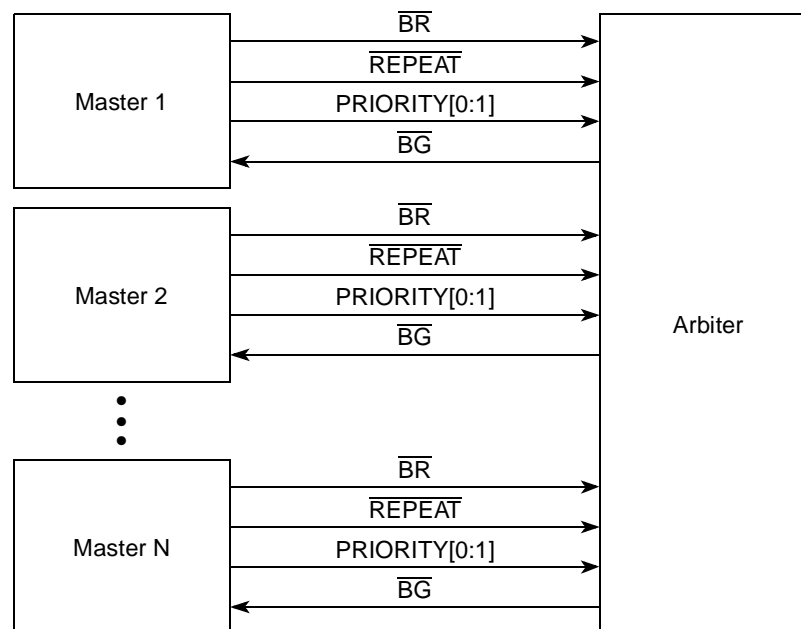


Figure 6-9. Address Bus Arbitration

A master has to acquire address bus ownership before it starts any transaction. The master asserts its own bus request signal along with the arbitration attribute signals  $\overline{\text{REPEAT}}$  and  $\text{PRIORITY}[0:1]$ . The arbiter later asserts the corresponding address bus grant signal to the requesting master depending on the system states and arbitration scheme. See [Section 6.3.1.1, “Address Bus Arbitration with  \$\text{PRIORITY}\[0:1\]\$ ,”](#) for details on arbitration scheme. When address bus grant is received the master can start the address tenure.

### 6.3.1.1 Address Bus Arbitration with $\text{PRIORITY}[0:1]$

Whenever a master asserts its bus request to acquire address bus ownership, it can drive its  $\text{PRIORITY}[0:1]$  signals to indicate request priority. The master would be served sooner because of its higher priority level. The arbiter takes this extra information into consideration in order to yield better service for a higher priority request than a lower priority request. Therefore, the arbiter operates according to the following priority-based arbitration scheme:

1. For every priority level a fair arbitration scheme is used (a simple round-robin scheme)
2. For every priority level other than 0, one place is reserved as a place-holder for lower level arbitration rings.
3. Each master can change its priority level at any time.

[Figure 6-10](#) shows an example of priority-based arbitration algorithm with four priority levels. In this example, if all masters request the bus continuously, the following order of bus grants occurs with the specific bandwidth:

- M6 gets  $\frac{1}{2}$  of the bus bandwidth
- M4 and M5 each gets  $\frac{1}{6}$  of the bus bandwidth
- M0 and M3 each gets  $\frac{1}{18}$  of the bus bandwidth
- M1 and M2 each gets  $\frac{1}{36}$  of the bus bandwidth

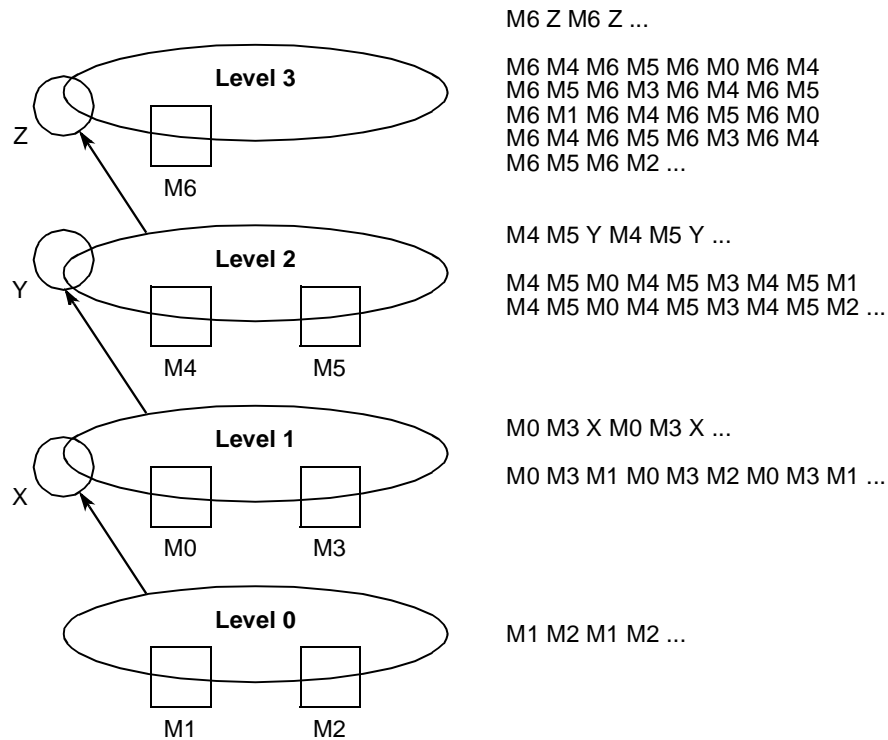


Figure 6-10. An Example of Priority-Based Arbitration Algorithm

#### NOTE

See each bus master's chapter and [Section 5.2.2.4, "System Priority and Configuration Register \(SPCR\),"](#) for more details about priority programming.

### 6.3.1.2 Address Bus Arbitration with $\overline{\text{REPEAT}}$

When a master owns the address bus and wants to perform another transaction, it can assert bus request along with  $\overline{\text{REPEAT}}$  to make a repeat request to the arbiter. Consequently, the arbiter asserts bus grant to the same master if the current address tenure is not being  $\overline{\text{ARTRY}}$ ed. This happens regardless of the priority level of bus request from other masters. In another word, "repeat request" overrides the priority scheme.

Even though repeat request can improve the page hit ratio and the overall memory bandwidth efficiency, it can increase the worst-case latency of individual master. Therefore, the arbiter has programmable counter to limit the maximum number of consecutive transactions that are performed by masters. Whenever the counter expires, the arbiter ignores the  $\overline{\text{REPEAT}}$  signal and falls back to the regular arbitration scheme.

### 6.3.1.3 Address Bus Arbitration after $\overline{\text{ARTRY}}$

The  $\overline{\text{ARTRY}}$  protocol is used primarily by the CPU to interrupt a transaction that hits to a modified line in its D-cache, so that it can maintain data coherency by performing the snoop copyback. When the CPU asserts  $\overline{\text{ARTRY}}$ , the bus is immediately granted to the CPU to perform snoop copyback. After the

completion of snoop copyback, the arbiter grants the bus back to the master that had its transaction  $\overline{\text{ARTRY}}$ ed.

#### 6.3.1.4 Address Bus Parking

The arbiter supports address bus parking. This feature implies that when no master is requesting the bus (all bus requests are negated), the arbiter can choose to park the address bus (or assert the address bus grant) to a master. The parked master can skip the bus request and assume the bus mastership directly. This reduces the access latency for parked master.

See [Section 6.2.1, “Arbiter Configuration Register \(ACR\),”](#) for more details about ACR[APARK] and ACR[PARKM].

#### 6.3.1.5 Data Bus Arbitration

For every committed address tenure a data tenure is required to complete the transaction.

In the device system, the arbiter controls the issuing of data bus grants to a master and a slave, which are involved in a data tenure of a previously performed address tenure.

### 6.3.2 Bus Error Detection

The arbiter is responsible for tracking the following cases on the bus:

- Address time out
- Data time out
- Transfer error
- Address only transaction type
- Reserved transaction type
- Illegal (**eciwx/ecowx**) transaction type

#### 6.3.2.1 Address Time Out

Address time out occurs, if the address tenure was not ended before the specified time-out period (programmed by ATR[ATO]). In this case, the arbiter performs as follows:

1. Ends the address tenure.
2. Starts data tenure and ends it by asserting transfer error.
3. Reports on the event to AER[ATO].
4. Issues reset request, MCP or regular interrupt according to AERR[ATO] and AIDR[ATO], if enabled by AMR[ATO].
5. Updates transaction attributes and address of AEATR and AEADR for the first error event.

### 6.3.2.2 Data Time Out

Data time out occurs, if the data tenure was not ended before the specified time-out period (programmed by ATR[DTO]). In this case, the arbiter performs as follows:

1. Ends the data tenure by asserting transfer error.
2. Reports on this event in AER[DTO].
3. Issues reset request, MCP or regular interrupt according to AERR[DTO] and AIDR[DTO], if enabled by AMR[DTO].
4. Updates transaction attributes and address of AEATR and AEADR for the first error event.

### 6.3.2.3 Transfer Error

The arbiter tracks the transfer error asserted by one of the slaves. In this case, the arbiter performs as follows:

1. Reports on the event to AER[ETEA].
2. Issues reset request, MCP or regular interrupt according to AERR[ETEA] and AIDR[ETEA] if enabled by AMR[ETEA].
3. Updates transaction attributes and address of AEATR and AEADR for the first error event.

### 6.3.2.4 Address Only Transaction Type

Table 6-10 shows transaction types that are defined as address only:

**Table 6-10. Address Only Transaction Type Encoding**

ttype[0:4]	Bus Commands
00000	Clean block
00100	Flush block
01000	Sync
01100	Kill block
10000	eieio
11000	TLB Invalidate
00001	lwarx reservation set
01001	tlbsync
01101	icbi

The arbiter allows address-only (AO) transactions on the bus, and the G2 core has ability to issue address-only (AO) transactions (see HID0 [ABE] in the *G2 PowerPC Core Reference Manual*, Rev 1). As there is no advantage in using AO transaction in this system, the bus monitor allows the detection of AO transactions and treats them as an error.

For the transaction with an address-only transfer type, the arbiter performs as follows:

1. Ends the address tenure by asserting  $\overline{AACK}$ .
2. Reports on the event to AER[AO].
3. Issues reset request, MCP or regular interrupt according to AERR[AO] and AIDR[AO] if enabled by AMR[AO].

4. Updates transaction attributes and address of AEATR and AEADR for the first error event.

### 6.3.2.5 Reserved Transaction Type

Table 6-11 shows transaction types defined as reserved.

**Table 6-11. Reserved Transaction Type Encoding**

ttype[0:4]	Bus Commands
00101	Reserved
1xx01	Reserved for customer
10110	Reserved
00011	Reserved
00111	Reserved
01111	Reserved
1xx11	Reserved for customer

For the transaction with a reserved transfer type, the arbiter performs as follows:

1. Ends the address tenure by asserting  $\overline{AACK}$ .
2. Reports on the event to AER[RES].
3. Issues reset request, MCP or regular interrupt according to AERR[RES] and AIDR[RES], if enabled by AMR[RES].
4. Updates transaction attributes and address of AEATR and AEADR for the first error event.

### 6.3.2.6 Illegal (eciwx/ecowx) Transaction Type

Table 6-12 shows transaction types defined as illegal.

**Table 6-12. Illegal Transaction Type Encoding**

ttype[0:4]	Bus command
10100	External control word write ( <b>ecowx</b> )
11100	External control word read ( <b>eciwx</b> )

For the transaction with an illegal (**eciwx**, **ecowx**) transfer type, the arbiter performs as follows:

1. Ends the address tenure by asserting  $\overline{AACK}$ .
2. Starts data tenure and ends data tenure by asserting  $\overline{TEA}$ .
3. Reports on the event in AER[ECW].
4. Issues reset request, MCP or regular interrupt according to AERR[ECW] and AIDR[ECW], if enabled by AMR[ECW].
5. Updates transaction attributes and address of AEATR and AEADR for the first error event.

For more information, see the following sections:

- Section 6.2.3, “Arbiter Event Register (AER)”
- Section 6.2.4, “Arbiter Interrupt Definition Register (AIDR)”
- Section 6.2.5, “Arbiter Mask Register (AMR)”
- Section 6.2.6, “Arbiter Event Attributes Register (AEATR)”

- Section 6.2.7, “Arbiter Event Address Register (AEADR)”
- Section 6.2.8, “Arbiter Event Response Register (AERR)”

## 6.4 Initialization/Applications Information

The following sections describe the initialization and error handling sequences for the arbiter.

### 6.4.1 Initialization Sequence

The following initialization sequence is recommended:

1. Write to ACR to configure pipeline depth, address bus parking mode, global maximum repeat count.
2. Write to AERR defines whether different error events cause a reset request or an interrupt.
3. Write to AIDR defines the kind of interrupt (regular or MCP) caused by each error event. Note that this is necessary only if interrupts are enabled and AERR defines error events to cause interrupt.
4. Write to AMR to enable interrupts.
5. Write to ATR to set the ATO and DTO timers. Note that this is only necessary if the required timers are less than the maximum value (which is default).

### 6.4.2 Error Handling Sequence

The following error handling sequence is recommended:

1. Read to AER to find out about the error that occurred in the system. Also, read the values of AEATR and AEADR to check on the first error event in the system.
2. If those registers are not accessible because of a bus deadlock situation, reset the chip and read the values of the AEATR and AEADR registers to check on the event that causes this problem to the system. Use  $\overline{\text{HRESET}}$  to reset the chip to guarantee that the information stored in AEATR and AEADR is not lost.
3. Clear all the previous events by writing ones to the AER. This register is also cleared after reset.





# Chapter 7

## e300 Processor Core Overview

This chapter provides an overview of features for the embedded microprocessors in the e300 core family, which are PowerPC microprocessors built on Power Architecture technology. Throughout this chapter, the terms 'e300 core', 'core', and 'processor' are used interchangeably. The term, 'e300c3' is used when describing an implementation-specific feature or when a difference exists between different configurations. The term 'e300' is used when describing a feature that pertains to the family of e300 processors. The MPC8308 uses an e300c3 core.

### 7.1 Overview

This section describes the details of the e300 core, provides a block diagram showing the major functional units, and briefly describes how these units interact. For additional information, see *e300 Power Architecture™ Core Family Reference Manual*.

The e300 core is a low-power implementation of this microprocessor family of reduced instruction set computing (RISC) microprocessors. The core implements the 32-bit portion of the PowerPC architecture, which defines 32-bit effective addresses, integer data types of 8, 16, and 32 bits, and floating-point data types of 32 and 64 bits.

The core is a superscalar processor that can issue and retire as many as three instructions per clock cycle. Instructions can execute out of program order for increased performance; however, the core makes completion appear sequential.

The e300 core integrates independent execution units including: an integer unit (IU), a floating-point unit (FPU), a branch processing unit (BPU), a load/store unit (LSU), and a system register unit (SRU). The e300c3 integrates an additional integer unit for a total of two IUs. The ability to execute instructions in parallel and the use of simple instructions with rapid execution times yield high efficiency and throughput for e300 core-based systems. Most integer instructions execute in one-clock cycle. The additional IUs along with enhanced multipliers in the e300c3 improve multiply instructions to a maximum two-cycle latency, a significant improvement from previous processors. In the e300c3 core, the FPU is pipelined so a single-precision multiply-add instruction can be issued and completed every clock cycle. The e300c3 core provide hardware support for all single- and double-precision floating-point operations for most value representations and all rounding modes.

Figure 7-1 shows a block diagram of the e300c3 core. Note that the e300c3 supports floating-point operations and includes two integer units.

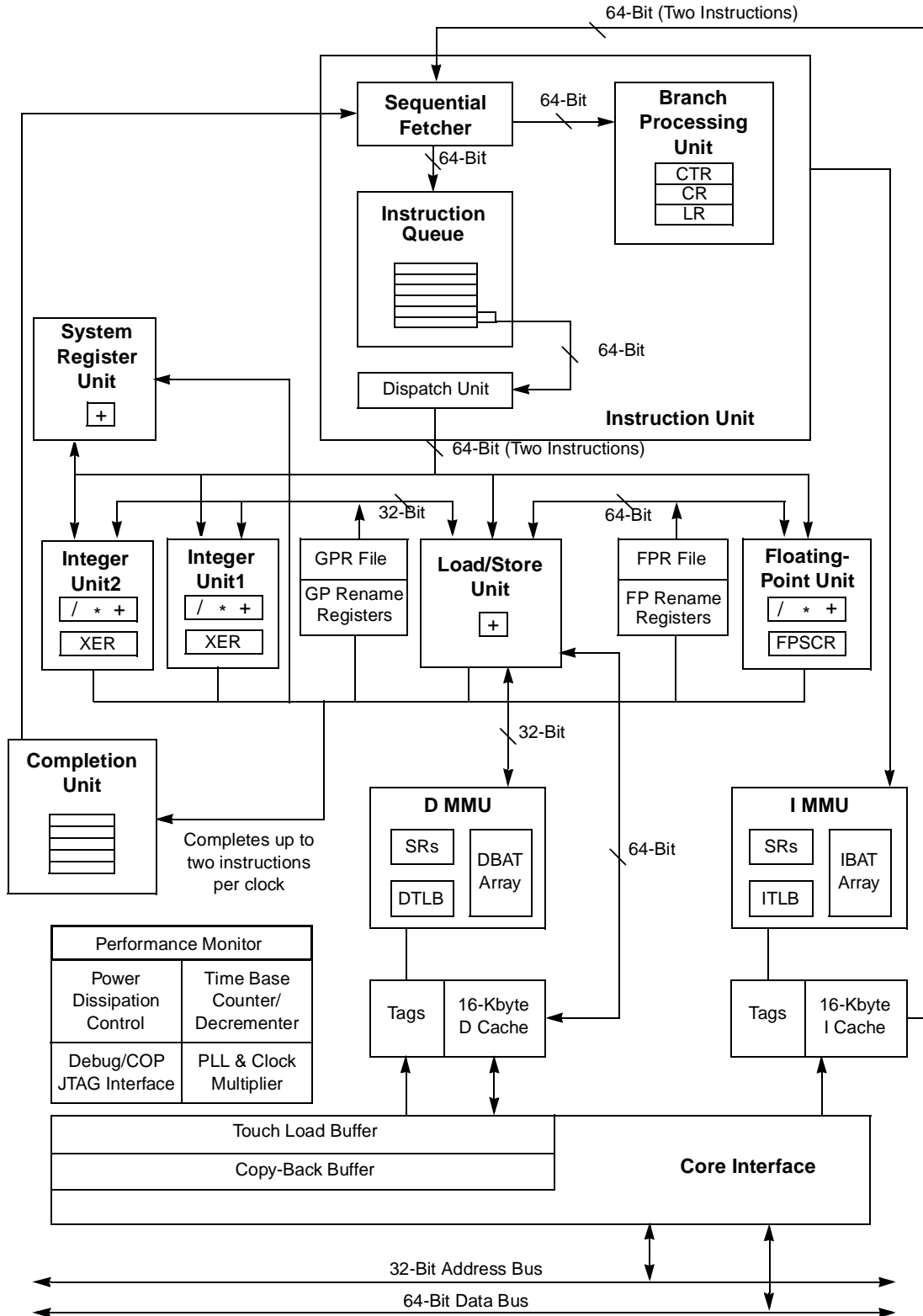


Figure 7-1. e300c3 Core Block Diagram

The e300c3 includes 16-Kbyte, four way set-associative instruction and data caches. The MMUs contain 64-entry, two-way, set-associative, data and instruction translation lookaside buffers (DTLB and ITLB) that provide support for demand-paged, virtual-memory, address translation, and variable-sized block translation. The TLBs use a least recently used (LRU) replacement algorithm and the caches use a pseudo least recently used algorithm (PLRU).

The core also supports block address translation through the use of two independent instruction and data block address translation (IBAT and DBAT) arrays, each containing eight pairs of BATs, an increase from four pairs of each type of BATs in the G2 core. This increase provides more flexibility in protecting accesses and providing translation on a segment, block, or page basis for memory accesses and I/O accesses. Effective addresses are compared simultaneously with all eight entries in the BAT array during block translation. In accordance with the PowerPC architecture, if an effective address hits in both the TLB and BAT array, the BAT translation takes priority.

As part of the coherent system bus (CSB), the e300 core has a 64-bit data bus and a 32-bit address bus. During normal operation, the e300 core provides a three-state (modified, exclusive, and invalid) coherency protocol which is a compatible subset of a four-state (modified/exclusive/shared/invalid) MESI protocol. However, the e300 data cache contains a programmable MESI extension that supports the shared cache coherency state (similar to other PowerPC processors). Both protocols operate coherently in systems that contain four-state caches. Although MESI is supported by the e300 core, it is not implemented on the MPC8308. The core also supports single-beat and burst data transfers for memory accesses and supports memory-mapped I/O operations.

The true little-endian mode is another enhanced capability of the e300 core. Unlike the PowerPC little-endian mode (which manipulates only the address bits), no longer supported on the e300, the true little-endian mode actually operates on true little-endian instructions and data from memory.

The critical interrupt is an additional interrupt in the e300 core and has higher priority order than the system management interrupt. Also, debug features are improved in the e300. Additional SPRG interrupt handling registers are provided for enhancing flexibility for the operating system.

The e300c3 include a performance monitor facility that provides the ability to monitor and count predefined events such as core clocks, misses in the instruction cache, data cache, types of instructions dispatched, mispredicted branches, and other occurrences. The count of such events (which may be an approximation) can be used to trigger the performance monitor interrupt. [Section 7.1.7.5, “Core Performance Monitor,”](#) describes the operation of the performance monitor diagnostic tool.

## 7.1.1 Features

This section describes the major features of the e300 core:

- High-performance, superscalar microprocessor core
  - As many as three instructions issued and retired per clock (two instructions plus one branch instruction)
  - As many as five instructions in execution per clock
  - Single-cycle execution for most instructions
  - Pipelined floating-point unit (FPU) for all single- and double-precision operations

- Independent execution units and two register files
  - Branch processing unit (BPU) featuring static branch prediction
  - Two 32-bit integer units (IU) in the e300c3
  - FPU based on the IEEE Std 754™ for both single- and double-precision operations
  - Load/store unit (LSU) for data transfer between data-cache and general-purpose registers (GPRs) and floating-point registers (FPRs)
  - System register unit (SRU) that executes condition register (CR), special-purpose register (SPR), and integer add/compare instructions. Add/compare instructions are also executed in the IUs.
  - Thirty-two 32-bit GPRs for integer operands
  - Thirty-two 64-bit FPRs for single- or double-precision operands
- High instruction and data throughput
  - Zero-cycle branch capability (branch folding)
  - Programmable static branch prediction on unresolved conditional branches
  - Two integer units with enhanced multipliers in the e300c3 for increased integer instruction throughput and a maximum two-cycle latency for multiply instructions
  - Instruction fetch unit capable of fetching two instructions per clock from the instruction cache
  - A six-entry instruction queue (IQ) that provides lookahead capability
  - Independent pipelines with feed-forwarding that reduces data dependencies in hardware
  - 16-Kbyte, four-way set-associative instruction and data caches on the e300c3
  - Cache write-back or write-through operation programmable on a per-page or per-block basis
  - Features for instruction and data cache locking and protection
  - BPU that performs CR lookahead operations
  - Address translation facilities for 4-Kbyte page size, variable block size, and 256-Mbyte segment size
  - A 64-entry, two-way, set-associative ITLB and DTLB
  - Eight-entry data and instruction BAT arrays providing 128-Kbyte to 256-Mbyte blocks
  - Software table search operations and updates supported through fast trap mechanism
  - 52-bit virtual address; 32-bit physical address
- Facilities for enhanced system performance
  - A 64-bit split-transaction internal data bus interface to the coherent system bus (CSB) with burst transfers
  - Support for one-level address pipelining on the CSB interface
  - True little-endian mode for compatibility with other true little-endian devices
  - Critical interrupt support
  - Hardware support for misaligned little-endian accesses
  - Configurable processor bus frequency multipliers as defined in the *PowerQUICC II Pro MPC8308 Hardware Specification*

- Integrated power management
  - Internal processor/bus clock multiplier ratios
  - Three power-saving modes: doze, nap, and sleep
  - Automatic dynamic power reduction when internal functional units are idle
- In-system testability and debugging features through JTAG boundary-scan capability

Features specific to the e300 core not present on the G2 processors follow:

- Enhancements to the register set
  - The e300 core has one more HID0 bit than the G2:
    - The enable cache parity checking (ECPE) bit, HID0[1], gives the e300 core the ability to enable the taking of a machine check interrupt based on the detection of a cache parity error
- Enhancements to cache implementation
  - 16-Kbyte, four-way set-associative instruction and data caches on the e300c3.
  - Full parity checking is performed on both instruction and data cache memory arrays
  - Lockable L1 instruction and data caches—entire cache or on a per-way basis up to 3 of 4 ways on the e300c3
  - New **icbt** instruction supports initialization of instruction cache
  - Data cache supports four-state MESI coherency protocol (not implemented on MPC8308)
  - The instruction cache is blocked only until the critical load completes (hit under reloads allowed)
  - Instruction cancel mechanism improves utilization of instruction cache by supporting hits-under-cancels and misses-under-cancels.
  - The critical double word is simultaneously written to the cache and forwarded to the requesting unit, thus minimizing stalls due to load delays.
  - Data cache queue sharing makes cast-outs and snoop pushes more efficient
  - Provides for an optional data cache operation broadcast feature (enabled by HID0[ABE]) that allows for coherent system management. All of the data cache control instructions, except **dcbz** (**dcbi**, **dcbf**, and **dcbst**) require that HID0[ABE] be enabled to broadcast.
  - Instruction fetch burst feature allows all instruction fetches from caching-inhibited space to be performed on the bus as burst transactions
- Interrupts
  - The e300 core offers hardware support for misaligned little-endian accesses. Little-endian load/store accesses that are not on a word boundary, except for strings and multiples, generate interrupts under the same circumstances as big-endian accesses.
  - The e300 core supports true little-endian mode to minimize the impact on software porting from true little-endian systems.
  - An input interrupt signal,  $\overline{cint}$ , is provided to trigger the critical interrupt exception on the e300 core. The pm\_event\_in input signal can be used by the performance monitor counters to trigger an interrupt upon overflow on the e300c3 .
- Bus clock—PLL configuration signals include seven signals for settings and control: *pll\_cfg[0:6]*.

- Debug features
  - Breakpoint status recorded in DBCR and IBCR control registers
  - Two signals for the debug interface:  $\overline{stopped}$  and  $\overline{ext\_halt}$
  - Performance monitor registers for system analysis in the e300c3

Figure 7-1 provides a block diagram of the e300 core that shows how the execution units—IU, FPU, BPU, LSU, and SRU—operate independently and in parallel. It should be noted that this is a conceptual diagram and does not attempt to show how these features are physically implemented on the device.

The e300 core provides address translation and protection facilities, including an ITLB, a DTLB, and instruction and data BAT arrays. Instruction fetching and issuing are handled in the instruction unit. Translation of addresses for cache or external memory accesses are handled by the MMUs. Both units are discussed in more detail in Section 7.1.2, “Instruction Unit,” and Section 7.1.5.1, “Memory Management Units (MMUs).”

## 7.1.2 Instruction Unit

As shown in Figure 7-1, the e300 core instruction unit, which contains a fetch unit, instruction queue, dispatch unit, and BPU, provides centralized control of instruction flow to the execution units. The instruction unit determines the address of the next instruction to be fetched based on information from the sequential fetcher and from the BPU.

The instruction unit fetches the instructions from the instruction cache into the instruction queue. The BPU receives branch instructions from the fetcher and uses static branch prediction to allow fetching from a predicted instruction stream while a conditional branch is evaluated. The BPU folds out for unconditional branch instructions and conditional branch instructions unaffected by instructions in the execution pipeline.

Instructions issued beyond a predicted branch cannot complete execution until the branch is resolved, preserving the programming model of sequential execution. If any of these are branch instructions, they are decoded but not issued. Instructions to be executed by the FPU, IU, LSU, and SRU are issued and allowed to progress up to the register write-back stage. Write-back is allowed when a correctly predicted branch is resolved, and execution continues along the predicted path.

If branch prediction is incorrect, the instruction unit flushes all predicted path instructions, and instructions are issued from the correct path.

### 7.1.2.1 Instruction Queue and Dispatch Unit

The instruction queue (IQ), shown in Figure 7-1, holds as many as six instructions and loads up to two instructions from the instruction unit during a single cycle. The instruction fetch unit continuously loads as many instructions as space in the IQ allows. Instructions are dispatched to their respective execution units from the dispatch unit at a maximum rate of two instructions per cycle. Dispatching is facilitated to the IUs, FPU, LSU, and SRU by the provision of a reservation station at each unit. The dispatch unit performs source and destination register dependency checking, determines dispatch serializations, and inhibits subsequent instruction dispatching as required.

For a more detailed overview of instruction dispatch, see Section 7.4.6, “Instruction Timing.”

### 7.1.2.2 Branch Processing Unit (BPU)

The BPU receives branch instructions from the fetch unit and performs CR lookahead operations on conditional branches to resolve them early, achieving the effect of a zero-cycle branch in many cases.

The BPU uses a bit in the instruction encoding to predict the direction of the conditional branch. Therefore, when an unresolved conditional branch instruction is encountered, the core fetches instructions from the predicted target stream until the conditional branch is resolved.

The BPU contains an adder to compute branch target addresses and three user-control registers: the link register (LR), the count register (CTR), and the conditional register (CR). The BPU calculates the return pointer for sub-routine calls and saves it into the LR for certain types of branch instructions. The LR also contains the branch target address for the Branch Conditional to Link Register (**bclr<sub>x</sub>**) instruction. The CTR contains the branch target address for the Branch Conditional to Count Register (**bcctr<sub>x</sub>**) instruction. The contents of the LR and CTR can be copied to or from any GPR. Because the BPU uses dedicated registers rather than GPRs or FPRs, execution of branch instructions is largely independent from execution of integer and floating-point instructions.

### 7.1.3 Independent Execution Units

The PowerPC architecture's support for independent execution units allows implementation of processors with out-of-order instruction execution. For example, because branch instructions do not depend on GPRs or FPRs, branches can often be resolved early, eliminating stalls caused by taken branches.

The four other execution units and the completion unit are described in the following sections.

#### 7.1.3.1 Integer Unit (IU)

The IU executes all integer instructions. The IU executes one integer instruction at a time, performing computations with its arithmetic logic unit (ALU), multiplier, divider, and XER register. Most integer instructions are single-cycle instructions. The 32 GPRs hold integer operands. Stalls due to contention for GPRs are minimized by the automatic allocation of rename registers. The core writes the contents of the rename registers to the appropriate GPR when integer instructions are retired by the completion unit. The e300c3 provides two integer units for greater integer instruction throughput along with enhanced multipliers in each IU for faster multiply-instruction execution.

#### 7.1.3.2 Floating-Point Unit (FPU)

The FPU contains a single-precision multiply-add array and the floating-point status and control register (FPSCR). The multiply-add array allows the core to efficiently implement multiply and multiply-add operations. The FPU is pipelined so that single- and double-precision instructions can be issued back-to-back. The 32 FPRs are provided to support floating-point operations. Stalls due to contention for FPRs are minimized by the automatic allocation of rename registers. The core writes the contents of the rename registers to the appropriate FPR when floating-point instructions are retired by the completion unit.

The e300c3 core supports all floating-point data types based on the IEEE 754 standard (normalized, denormalized, NaN, zero, and infinity) in hardware, eliminating the latency incurred by software interrupt routines.

### 7.1.3.3 Load/Store Unit (LSU)

The LSU executes all load and store instructions and provides the data transfer interface between the GPRs, FPRs, and the cache/memory subsystem. The LSU calculates effective addresses, performs data alignment, and provides sequencing for load/store string and multiple instructions.

Load and store instructions are issued and executed in program order; however, the memory accesses can occur out of order. Synchronizing instructions are provided to enforce strict ordering.

Cacheable loads, when free of data bus dependencies, can execute out of order with a maximum throughput of one per cycle and with a two-cycle total latency. Data returned from the cache is held in a rename register until the completion logic commits the value to a GPR or FPR. Stores cannot be executed in a predicted manner and are held in the store queue until the completion logic signals that the store operation is to be completed to memory. The core executes store instructions with a maximum throughput of one per cycle and with a three-cycle total latency. The time required to perform the actual load or store depends on whether the operation involves the cache, system memory, or an I/O device.

### 7.1.3.4 System Register Unit (SRU)

The SRU executes various system-level instructions, including condition register logical operations and move to/from special-purpose register instructions. It also executes integer add/compare instructions. In order to maintain system state, most instructions executed by the SRU are completion-serialized; that is, the instruction is held for execution in the SRU until all prior instructions issued have completed. Results from completion-serialized instructions executed by the SRU are not available or forwarded for subsequent instructions until they complete.

## 7.1.4 Completion Unit

The completion unit tracks instructions in program order from dispatch through execution and then completes. Completing an instruction commits the core to any architectural register changes caused by that instruction. In-order completion ensures the correct architectural state when the core must recover from a mispredicted branch or an interrupt.

Instruction state and other information required for completion is kept in a five-entry FIFO completion queue. A single completion queue entry is allocated for each instruction once it enters the execution unit from the dispatch unit. An available completion queue entry is a required resource for dispatch; if no completion entry is available, dispatch stalls. A maximum of two instructions per cycle are completed in order from the queue.

## 7.1.5 Memory Subsystem Support

The core provides separate instruction and data caches and MMUs. The core also provides an efficient processor bus interface to facilitate access to main memory and other bus subsystems. The memory subsystem support functions are described in the following sections.



### 7.1.5.1 Memory Management Units (MMUs)

The core MMUs support up to 4 Petabytes ( $2^{52}$ ) of virtual memory and 4 Gigabytes ( $2^{32}$ ) of physical memory (referred to as real memory in the architecture specification) for instruction and data. The MMUs also control access privileges for these spaces on block and page granularities. Referenced and changed status is maintained by the processor for each page to assist implementation of a demand-paged virtual memory system. Note that software assistance is required for the device to maintain reference and changed status. A key bit is implemented to provide information about memory protection violations prior to page table search operations.

The LSU calculates effective addresses for data loads and stores, performs data alignment to and from cache memory, and provides the sequencing for load and store string and multiple word instructions. The instruction unit calculates effective addresses for instruction fetching.

After an EA is generated, its higher-order bits are translated by the appropriate MMU into physical address bits. The lower-order EA bits are the same on the physical address which are directed to the on-chip cache and formed the index into a four-way set-associative tag array. After translating the address, the MMU passes the higher-order physical address bits to the cache and the cache lookup completes. For caching-inhibited accesses or accesses that miss in the cache, the untranslated lower-order address bits are concatenated with the translated higher-order address bits; the resulting 32-bit physical address is then used by the memory unit and the core interface to access external memory.

The MMU also directs the address translation and enforces the protection hierarchy programmed by the operating system in relation to the supervisor/user privilege level of the access and in relation to whether the access is a load or store.

For instruction fetches, the IMMU looks for the address in the ITLB and in the IBAT array. If an address hits both, the IBAT array translation is used. Data accesses cause a lookup in the DTLB and DBAT array. In most cases, the translation is in a TLB and the physical address bits are available to the on-chip cache.

The e300 core implements four more IBAT and four more DBAT entries than the G2.

When the EA misses in the TLBs, the core provides hardware assistance for software to perform a search of the translation tables in memory. The hardware assist consists of the following features:

- Automatic storage of the missed effective address in IMISS and DMISS
- Automatic generation of the primary and secondary hashed real addresses of the page-table entry group (PTEG), which are readable from the HASH1 and HASH2 register locations.  
The HASH data is generated from the contents of the IMISS or DMISS register. The register that is selected depends on the miss (instruction or data) that was last acknowledged.
- Automatic generation of the first word of the page table entry (PTE) of the tables being searched
- A real page address (RPA) register that matches the format of the lower word of the PTE
- TLB access instructions (**tlbli** and **tblld**) that are used to load an address translation into the instruction or data TLBs
- Shadow registers for GPR0–GPR3 that allow miss code to execute without corrupting the state of any of the existing GPRs. Shadow registers are used only for servicing a TLB miss.

For more information about memory management for the core, see [Section 7.4.5.2, “Implementation-Specific Memory Management.”](#)

### 7.1.5.2 Cache Units

The e300c3 provides 16-Kbyte, four-way set-associative instruction and data caches. The cache block is 32 bytes long. The caches adhere to a write-back policy, but the e300 core allows control of cacheability, write policy, and memory coherency at the page and block levels. The caches use a pseudo LRU replacement policy.

As shown in [Figure 7-1](#), the caches provide a 64-bit interface to the instruction fetch unit and LSU. The surrounding logic selects, organizes, and forwards the requested information to the requesting unit. Write operations to the cache can be performed on a byte basis, and a complete read-modify-write operation to the cache can occur in each cycle.

The load/store and instruction fetch units provide the caches with the address of the data or instruction to be fetched. In the case of a cache hit, the cache returns two words to the requesting unit.

Because the data cache tags are single-ported, simultaneous load/store and snoop accesses cause resource contention. Snoop accesses have the highest priority and are given first access to the tags, unless the snoop access coincides with a tag write; in this case the snoop is retried and must rearbitrate for cache access. Loads or stores deferred due to snoop accesses are performed on the clock cycle following the snoop.

The e300 core includes a new instruction cancel extension. The instruction cancel extension improves utilization of the instruction cache during cancel operations. It allows a new instruction fetch to be issued to the cache or to the bus if a canceled instruction fetch is pending or active on the bus. This supports hit-under-cancel and miss-under-cancel instruction fetch operations.

### 7.1.6 Bus Interface Unit (BIU)

Because the caches are on-chip, write-back caches, the most common transactions are burst-read memory operations, burst-write memory operations, and single-beat (noncacheable or write-through) memory read and write operations. There can also be address-only operations, variants of the burst and single-beat operations (for example, global memory operations that are snooped and atomic memory operations), and address retry activity (for example, when a snooped read access hits a modified cache block).

Memory accesses can occur in single-beat (1–8 bytes) and four-beat burst (32 bytes) data transfers on the 64-bit data bus. The address and data buses operate independently to support pipelining and split transactions during memory accesses.

The e300 bus interface unit (BIU) has been enhanced to allow a pipeline slot to become available once a previous transaction has been granted the data bus (that is, as early as when the data tenure starts rather than after the data tenure completes), thus allowing for greater bus utilization in systems that support it. This is sometimes referred to as 1 1/2-level pipelining.

Typically, memory accesses are weakly ordered, meaning that sequences of operations, including load/store string and multiple instructions, do not necessarily complete in the order they begin. This weak ordering maximizes the efficiency of the bus without sacrificing coherency of the data. The core allows read operations to precede store operations (except when a dependency exists, or in cases where a

noncacheable access is performed), and provides support for a write operation to proceed a previously queued read data tenure (for example, allowing a snoop push to be enveloped by the address and data tenures of a read operation). Because the processor can dynamically optimize run-time ordering of load/store traffic, overall performance is improved.

## 7.1.7 System Support Functions

The e300 core implements several support functions that include power management, time base/decrementer registers for system timing tasks, a JTAG (based on IEEE Std 1149.1™) interface, hardware debug, and a phase-locked loop (PLL) clock multiplier. These system support functions are described in the following sections.

### 7.1.7.1 Power Management

The e300 core provides four power modes, selectable by setting the appropriate control bits in the machine state register (MSR) and the hardware implementation register 0 (HID0). When entering into a power mode other than full-power, the core requests entry via a  $\overline{qreq}$  signal and enters another power mode after an acknowledge ( $\overline{qack}$ ) is received. The four power modes are as follows:

- Full-power
 

This is the default power state of the e300 core. The e300 core is fully powered and the internal functional units are operating at the full processor clock speed. If the dynamic power management mode is enabled, functional units that are idle automatically enters a low-power state without affecting performance, software execution, or external hardware.
- Doze
 

All the functional units of the e300 core are disabled except for the time base/decrementer registers and the bus snooping logic. When the processor is in doze mode, an external asynchronous interrupt, system management interrupt, decrementer interrupt, hard or soft reset, or machine check brings the e300 core into the full-power state. The core in doze mode maintains the PLL in a fully-powered state and locked to the system external clock input ( $sysclk$ ), so a transition to the full-power state takes only a few processor clock cycles.
- Nap
 

The nap mode further reduces power consumption by disabling bus snooping, leaving only the time base register and the PLL in a powered state. The core returns to the full-power state on receipt of an external asynchronous interrupt, system management interrupt, decrementer interrupt, hard or soft reset, or machine check input ( $\overline{mcp}$ ) signal. A return to full-power state from a nap state takes only a few processor clock cycles.
- Sleep
 

Sleep mode reduces power consumption to a minimum by disabling all internal functional units; then external system logic may disable the PLL and  $sysclk$ . Returning the core to the full-power state requires the enabling of the PLL and  $sysclk$ , followed by the assertion of an external asynchronous interrupt, system management interrupt, hard or soft reset, or  $\overline{mcp}$  signal after the time required to relock the PLL.

### 7.1.7.2 Time Base/Decrementer

The time base is a 64-bit register (accessed as two 32-bit registers) that is incremented once every four bus clock cycles; external control of the time base is provided through the time base/decrementer clock base enable (*tben*) signal. The decrementer is a 32-bit register that generates a decrementer interrupt after a programmable delay. The contents of the decrementer register are decremented once every four bus clock cycles, and the decrementer interrupt is generated as the count passes through zero.

### 7.1.7.3 JTAG Test and Debug Interface

The core provides JTAG and hardware debug functions for facilitating board testing and chip debugging. The JTAG test interface (based on IEEE 1149.1) provides a means for boundary-scan testing of the core and the attached system logic. The hardware debug function accesses the JTAG test port, providing a means for executing test routines and facilitating chip and software debugging.

All instruction and data address breakpoints are accessible in the IBCR and DBCR. See [Section 7.4.8, “Debug Features,”](#) for more information.

### 7.1.7.4 Clock Multiplier

The internal clocking of the e300 core is generated from and synchronized to the external clock signal, *sysclk*, by means of a voltage-controlled, oscillator-based PLL. The PLL provides programmable internal processor clock multiplier ratios which multiply the externally supplied clock frequency. The bus clock is the same frequency and is synchronous with *sysclk*. The configuration of the PLL can be read by software from the hardware implementation register 1 (HID1).

### 7.1.7.5 Core Performance Monitor

The performance monitor provides the ability to count predefined events and processor clocks associated with particular operations, such as cache misses, mispredicted branches, or the number of cycles an execution unit stalls. The count of such events can be used to trigger the performance monitor interrupt.

The performance monitor can be used to do the following:

- Improve system performance by monitoring software execution and then recoding algorithms for more efficiency. For example, memory hierarchy behavior can be monitored and analyzed to optimize task scheduling or data distribution algorithms.
- Characterize processors in environments not easily characterized by benchmarking.
- Help system developers bring up and debug their systems.

The performance monitor uses the following resources:

- The performance monitor mark bit in the MSR (MSR[PMM]). This bit controls which programs are monitored.
- The move to/from performance monitor registers (PMR) instructions, **mtpmr** and **mfpmr**.
- The external core input, *pm\_event\_in*.
- PMRs

- The performance monitor counter registers (PMC0–PMC3) are 32-bit counters used to count software-selectable events. Each counter counts up to 128 events. UPMC0–UPMC3 provide user-level read access to these registers. They are identified in [Table 7-2](#).
- The performance monitor global control register (PMGC0) controls the counting of performance monitor events. It takes priority over all other performance monitor control registers. UPMGC0 provides user-level read access to PMGC0.
- The performance monitor local control registers (PMLCa0–PMLCa3) control each individual performance monitor counter. Each counter has a corresponding PMLCa register. UPMLCa0–UPMLCa3 provide user-level read access to PMLCa0–PMLCa3).
- The performance monitor interrupt is assigned to interrupt vector 0x0F00.

Software communication with the performance monitor is achieved through PMRs rather than SPRs. The PMRs are used for enabling conditions that can trigger the performance monitor interrupt.

## 7.2 e300 Processor and System Version Numbers

[Table 7-1](#) lists the revision codes in the processor version register (PVR) and the system version register (SVR) to the revision level marked on the device. These registers can be accessed as SPRs through the e300 core (see [Figure 7-2](#)).

**Table 7-1. Device Revision Level Cross-Reference**

MPC8308 Revision	Processor Version Register (PVR)	System Version Register (SVR)
A	8085_0020	8101_0110

## 7.3 PowerPC Architecture Implementation

The PowerPC architecture consists of the following layers, and adherence to the PowerPC architecture can be measured in terms of which of the following levels of the architecture is implemented:

- User instruction set architecture (UISA)  
Defines the base user-level instruction set, user-level registers, data types, floating-point interrupt model, memory models for a uniprocessor environment, and programming model for a uniprocessor environment.
- Virtual environment architecture (VEA)  
Describes the memory model for a multiprocessor environment, defines cache control instructions, and describes other aspects of virtual environments. Implementations that conform to the VEA also adhere to the UISA but may not necessarily adhere to the OEA.
- Operating environment architecture (OEA)  
Defines the memory management model, supervisor-level registers, synchronization requirements, and interrupt model. Implementations that conform to the OEA also adhere to the UISA and VEA.

The PowerPC architecture allows a wide range of designs for such features as cache and core interface implementations.

## 7.4 Implementation-Specific Information

This section describes the PowerPC architecture in general and specific details about the implementation of the e300 core as a low-power, 32-bit member of this PowerPC core family. The main topics addressed are as follows:

- [Section 7.4.1, “Register Model,”](#) describes the registers for the operating environment architecture common among e300 cores that implement the PowerPC architecture and describes the programming model. It also describes the additional registers that are unique to the core.
- [Section 7.4.2, “Instruction Set and Addressing Modes,”](#) describes the PowerPC instruction set and addressing modes for the OEA, and defines and describes the instructions implemented in the core.
- [Section 7.4.3, “Cache Implementation,”](#) describes the cache model that is defined generally for cores that implement the PowerPC architecture by the VEA. It also provides specific details about the e300 core cache implementation.
- [Section 7.4.4, “Interrupt Model,”](#) describes the interrupt model of the OEA and the differences in the core interrupt model.
- [Section 7.4.5, “Memory Management,”](#) describes generally the conventions for memory management among these cores. This section also describes the core implementation of the 32-bit PowerPC memory management specification.
- [Section 7.4.6, “Instruction Timing,”](#) provides a general description of the instruction timing provided by the superscalar, parallel execution supported by the PowerPC architecture and the e300 core.
- [Section 7.1.6, “Bus Interface Unit \(BIU\),”](#) describes the signals implemented on the core.

The e300 core is a high-performance, superscalar processor core. The PowerPC architecture allows optimizing compilers to schedule instructions to maximize performance through efficient use of the PowerPC instruction set and register model. The multiple, independent execution units allow compilers to optimize instruction throughput. Compilers that take advantage of the flexibility of the PowerPC architecture can additionally optimize system performance.

The following sections summarize the features of the core, including both those that are defined by the architecture and those that are unique to the various core implementations.

Specific features of the core are listed in [Section 7.1.1, “Features.”](#)

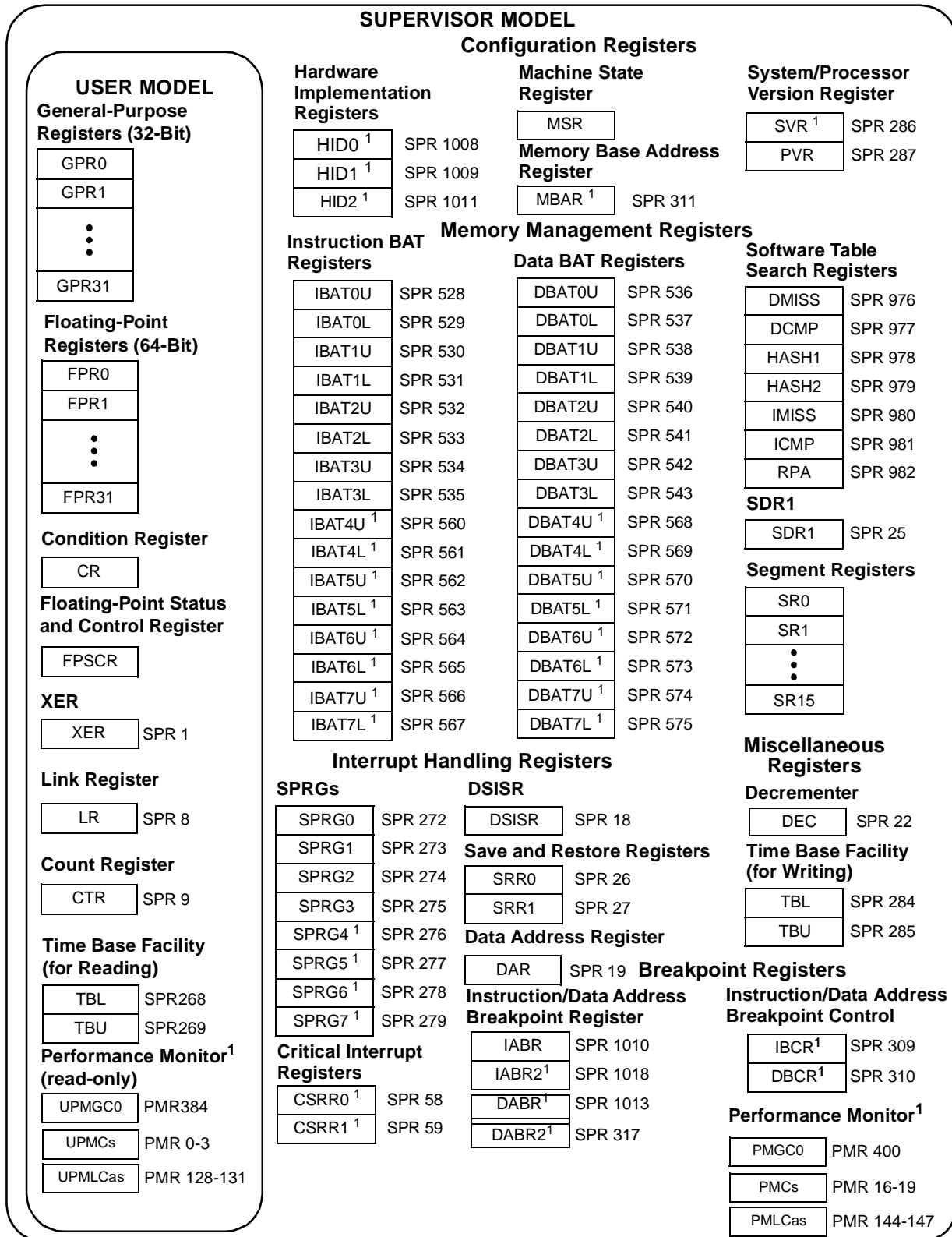
### 7.4.1 Register Model

The PowerPC architecture defines register-to-register operations for most computational instructions. Source operands for these instructions are accessed from the registers or are provided as immediate values embedded in the instruction opcode. The three-register instruction format allows specification of a target register distinct from the two-source operands. Load and store instructions transfer data between registers and memory.

The e300 core has two levels of privilege: supervisor mode of operation (typically used by the operating system) and user mode of operation (used by the application software). The programming models incorporate 32 GPRs, 32 FPRs, special-purpose registers (SPRs), and several miscellaneous registers. Each core also has its own unique set of hardware implementation (HID) registers.

Having access to privileged instructions, registers, and other resources allows the operating system to control the application environment (providing virtual memory and protecting operating system and critical machine resources). Instructions that control the state of the e300 core, the address translation mechanism, and supervisor registers can be executed only when the core is operating in supervisor mode.

Figure 7-2 shows all the core registers available at the user and supervisor level. The numbers to the right of the SPRs indicate the number that is used in the syntax of the instruction operands for the move to/from SPR instructions.



<sup>1</sup> These registers are e300 core implementation-specific (not defined by the PowerPC architecture).

**Figure 7-2. e300 Programming Model—Registers**



The following sections describe the e300 core-implementation-specific features as they apply to registers.

### 7.4.1.1 UISA Registers

UISA registers are user-level registers that include the following.

#### 7.4.1.1.1 General-Purpose Registers (GPRs)

The PowerPC architecture defines 32 user-level GPRs that are 32 bits wide in 32-bit cores. The GPRs serve as the data source or destination for all integer instructions.

#### 7.4.1.1.2 Floating-Point Registers (FPRs)

The PowerPC architecture also defines 32 user-level, 64-bit FPRs. The FPRs serve as the data source or destination for floating-point instructions. These registers can contain data objects of either single- or double-precision floating-point formats.

#### 7.4.1.1.3 Condition Register (CR)

The CR is a 32-bit user-level register that provides a mechanism for testing and branching. It consists of eight 4-bit fields that reflect the results of certain operations, such as move, integer and floating-point comparisons, arithmetic, and logical operations.

#### 7.4.1.1.4 Floating-Point Status and Control Register (FPSCR)

The user-level FPSCR contains all floating-point exception signal bits, exception summary bits, exception enable bits, and rounding control bits needed for compliance with the IEEE 754 standard.

#### 7.4.1.1.5 User-Level SPRs

The PowerPC architecture defines numerous special purpose registers that serve a variety of functions, such as providing controls, indicating status, configuring the core, and performing special operations. During normal execution, a program can access the registers, as shown in [Figure 7-2](#), depending on the program's access privilege (supervisor or user, determined by the privilege-level bit, MSR[PR]). Note that GPRs and FPRs are accessed through operands that are part of the instructions. Access to registers can be explicit (that is, through the use of specific instructions for that purpose such as Move to Special-Purpose Register (**mtspr**) and Move from Special-Purpose Register (**mfspr**) instructions) or implicit, as the part of the execution of an instruction. Some registers are accessed both explicitly and implicitly. In the e300 core, all SPRs are 32 bits wide.

The following SPRs are accessible by user-level software:

- Link register (LR)  
The LR can be used to provide the branch target address and to hold the return address after branch and link instructions. The LR is 32 bits wide in 32-bit implementations.
- Count register (CTR)  
The CTR is decremented and tested automatically as a result of branch-and-count instructions. The CTR is 32 bits wide in 32-bit implementations.

- XER register

The 32-bit XER contains the summary overflow bit, integer carry bit, overflow bit, and a field specifying the number of bytes to be transferred by a Load String Word Indexed (**lswx**) or Store String Word Indexed (**stswx**) instruction.

### 7.4.1.2 VEA Registers

The VEA introduces the time base facility (TB) for reading. The TB is a 64-bit register pair whose contents are incremented once every four core input clock cycles. The TB consists of two 32-bit registers—time base upper (TBU) and time base lower (TBL). Note that the time base registers are read-only in user state.

### 7.4.1.3 OEA Registers

OEA registers are supervisor-level registers that include the following.

#### 7.4.1.3.1 Machine State Register (MSR)

The MSR is a supervisor-level register that defines the state of the core. The contents of this register are saved when an interrupt is taken, and restored when the interrupt handling completes. A critical interrupt is taken in the e300 core when the  $\overline{cint}$  signal is asserted and MSR[CE] is set. The e300 core implements the MSR as a 32-bit register.

Table 7-2 shows the bit definitions for MSR.

**Table 7-2. MSR Bit Descriptions**

Bits	Name	Description
0 <sup>1</sup>	—	Reserved. Full function.
1–4 <sup>1</sup>	—	Reserved. Partial function.
5–9 <sup>1</sup>	—	Reserved. Full function.
10–12 <sup>1</sup>	—	Reserved. Partial function.
13	POW	Power management enable (implementation-specific) 0 Disables programmable power modes (normal operation mode) 1 Enables programmable power modes (nap, doze, or sleep mode). This bit controls the programmable power modes only; it has no effect on dynamic power management (DPM). MSR[POW] may be altered with an <b>mtmsr</b> instruction only. Also, when altering the POW bit, software may alter only this bit in the MSR and no others. The <b>mtmsr</b> instruction must be followed by a context-synchronizing instruction.
14	TGPR	Temporary GPR remapping (implementation-specific) 0 Normal operation 1 TGPR mode. GPR0–GPR3 are remapped to TGPR0–TGPR3 for use by TLB miss routines. The contents of GPR0–GPR3 remain unchanged while MSR[TGPR] = 1. Attempts to use GPR4–GPR31 with MSR[TGPR] = 1 yield undefined results. Temporarily replaces TGPR0–TGPR3 with GPR0–GPR3 for use by TLB miss routines. The TGPR bit is set when either an instruction TLB miss, data read miss, or data write miss interrupt is taken. The TGPR bit is cleared by an <b>rfi</b> instruction.
15	ILE	Interrupt little-endian mode. When an interrupt occurs, this bit is copied into MSR[LE] to select the endian mode for the context established by the interrupt.

Table 7-2. MSR Bit Descriptions (continued)

Bits	Name	Description
16	EE	External interrupt enable 0 The processor ignores external interrupts, system management interrupts, and decremter interrupts. 1 The processor is enabled to take an external interrupt, system management interrupt, or decremter interrupt.
17	PR	Privilege level 0 The processor can execute both user- and supervisor-level instructions 1 The processor can only execute user-level instructions
18	FP	Floating-point available 0 The processor prevents dispatch of floating-point instructions, including floating-point loads, stores, and moves. 1 The processor can execute floating-point instructions and can take floating-point enabled exception type program interrupts.
19	ME	Machine check enable 0 Machine check interrupts are disabled 1 Machine check interrupts are enabled
20	FE0	Floating-point exception mode 0
21	SE	Single-step trace enable 0 The processor executes instructions normally 1 The processor generates a trace interrupt upon the successful completion of the next instruction
22	BE	Branch trace enable 0 The processor executes branch instructions normally 1 The processor generates a trace interrupt upon the successful completion of a branch instruction
23	FE1	Floating-point exception mode 1
24	CE	Critical interrupt enable 0 Critical interrupts disabled 1 Critical interrupts enabled; critical interrupt and <b>rfci</b> instruction enabled The critical interrupt is an asynchronous implementation-specific interrupt. The critical interrupt vector offset is 0x00A00. The <b>rfci</b> instruction is implemented to return from these interrupt handlers. Also, CSRR0 and CSRR1 are used to save and restore the processor state for critical interrupts.
25	IP	Interrupt prefix. The setting of this bit specifies whether an interrupt vector offset is prepended with Fs or 0s. In the following description, <i>nnnnn</i> is the offset of the interrupt. 0 Interrupts are vectored to the physical address 0x000n_nnnn 1 Interrupts are vectored to the physical address 0xFFFFn_nnnn
26	IR	Instruction address translation 0 Instruction address translation is disabled 1 Instruction address translation is enabled
27	DR	Data address translation 0 Data address translation is disabled 1 Data address translation is enabled
28	—	Reserved. Full function.
29	PMM	Performance monitor mark bit (e300c3). System software can set PMM when a marked process is running to enable statistics to be gathered only during the execution of the marked process. MSR[PR] and MSR[PMM] together define a state that the processor (supervisor or user) and the process (marked or unmarked) may be in at any time. If this state matches an individual state specified in the PMLCan, the state for which monitoring is enabled, counting is enabled.

Table 7-2. MSR Bit Descriptions (continued)

Bits	Name	Description
30	RI	Recoverable interrupt (for system reset and machine check interrupts) 0 Interrupt is not recoverable 1 Interrupt is recoverable
31	LE	Little-endian mode enable 0 The processor runs in big-endian mode 1 The processor runs in little-endian mode.

<sup>1</sup> All reserved bits should be set to zero for future compatibility.

### 7.4.1.3.2 Segment Registers (SRs)

For memory management, 32-bit processors implement sixteen 32-bit SRs. To speed access, the core implements the SRs as two arrays: a main array for data memory accesses and a shadow array for instruction memory accesses. Loading a segment entry with the Move to Segment Register (**mtsr**) instruction loads both arrays.

### 7.4.1.3.3 Supervisor-Level SPRs

The e300 core, like the G2\_LE core, has additional supervisor-level SPRs, which are shown in [Figure 7-3](#). Two critical interrupt SPRs (CSRR0 and CSRR1), eight SPRGs (SPRG0–SPRG7), eight pairs of instruction BATs (IBAT0–IBAT7), eight pairs of data BATs (DBAT0–DBAT7), one system version register (SVR), one system memory base address (MBAR), one instruction address breakpoint control (IBCR), one data address breakpoint control (DBCR), a new instruction breakpoint register (IABR2), and two data address breakpoint registers (DABR and DABR2) are integrated into the core.

The following list discusses the supervisor-level SPRs.

- The DSISR defines the cause of data access and alignment interrupts. The cause of a DSI interrupt for a data breakpoint (match with DABR and DABR2) can be determined by the value of the DSISR[DABR] bit (bit 9).
- The data address register (DAR) holds the address of an access after an alignment or DSI interrupt. For example, it contains the address of the breakpoint match condition.
- The decremter register (DEC) is a 32-bit decrementing counter that provides a mechanism for causing a decremter interrupt after a programmable delay.
- SDR1 specifies the page table format used in virtual-to-physical address translation for pages. (Note that physical address is referred to as ‘real address’ in the architecture specification.)
- The machine status save/restore register 0 (SRR0) is used for saving the address of the instruction that caused the interrupt, and the address to return to when a Return from Interrupt (**rfi**) instruction is executed.
- The machine status save/restore register 1 (SRR1) is used to save machine status on interrupts and to restore machine status when an **rfi** instruction is executed.
- The SPRG0–SPRG7 registers are provided for operating system use. They reduce the latency that may be incurred in the saving of registers to memory while in a handler. Note that the e300 implements four more SPRGs than the G2 (SPRG0–SPRG3).

- The time base register (TB) is a 64-bit register that maintains the time of day and operates interval timers. It consists of two 32-bit fields: time base upper (TBU) and time base lower (TBL).
- The processor version register (PVR) is a read-only register that identifies the version (model) and revision level of the processor. See [Table 7-9](#) for the version and revision level of the PVR for the e300 processor core.
- The PowerPC architecture defines 16 block address translation (BAT) registers. The e300 core includes a total of eight pairs of DBAT and eight pairs of IBAT registers. See [Figure 7-2](#) for a list of the SPR numbers for the BAT arrays.

The following supervisor-level SPRs are implementation-specific (not defined in the PowerPC architecture):

- DMISS and IMISS are read-only registers that are loaded automatically on an instruction or data TLB miss.
- HASH1 and HASH2 contain the physical addresses of the primary and secondary page table entry groups (PTEGs).
- ICMP and DCMP contain a duplicate of the first word in the page table entry (PTE) for which the table search is looking.
- The required physical address (RPA) register is loaded by the core with the second word of the correct PTE during a page table search.
- The system version register (SVR) is available on the e300 core, which identifies the specific version (model) and revision level of the system-on-a-chip (SOC) integration.
- System memory base address (MBAR) is an implementation-specific register available on the e300 core. It supports a temporary storage for the system-level memory map.
- The instruction and data address breakpoint registers (IABR, IABR2, DABR, and DABR2) are loaded with an instruction or data address, respectively, that is compared to instruction addresses in the dispatch queue or to the data address in the LSU. When an address match occurs, a breakpoint interrupt is generated.
- One instruction breakpoint control register (IBCR) and one data breakpoint control register (DBCR) are implemented in the e300 core.
- To support critical interrupts, two registers (CSRR0 and CSRR1) are included in the e300 core.
- Eight SPRG registers (SPRG0–SPRG7) are in the e300 core.
- Block address translation (BAT) arrays—The e300 core has eight instruction and eight data BAT registers.
- The hardware implementation (HID0 and HID1) registers provide the means for enabling core checkstops and features and allow software to read the configuration of the PLL configuration signals. The HID2 register enables the true little-endian mode, cache way-locking, and the additional BAT registers.

Table 7-3 shows the bit definitions for HID0.

**Table 7-3. e300 HID0 Bit Descriptions**

Bits	Name	Function
0	EMCP	Enable $\overline{mcp}$ . The purpose of this bit is to mask out machine check interrupts caused by assertion of $\overline{mcp}$ , similar to how MSR[EE] can mask external interrupts. 0 Masks $\overline{mcp}$ . Asserting $\overline{mcp}$ does not generate a machine check interrupt or a checkstop. 1 Asserting $\overline{mcp}$ causes checkstop if MSR[ME] = 0 or a machine check interrupt if ME = 1
1	ECPE	Enable cache parity errors. 0 Disables instruction and data cache parity error reporting 1 Allows a detected cache parity error to cause a machine check interrupt if MSR[ME] = 1 or a checkstop if MSR[ME] = 0
2	EBA	Enable $\overline{ap\_in[0:3]}$ and $\overline{ape}$ for address parity checking. 0 Disables address parity checking during a snoop operation 1 Allows an address parity error during snoop operations to cause a checkstop if MSR[ME] = 0 or a machine check interrupt if MSR[ME] = 1
3	EBD	Enable $\overline{ape}$ for data parity checking. 0 Disables data parity checking 1 Allows a data parity error during reads to cause a checkstop if MSR[ME] = 0 or a machine check interrupt if MSR[ME] = 1
4	SBCLK	$clk\_out$ output enable. Used in conjunction with HID0[ECLK] and $\overline{hreset}$ to configure $clk\_out$ . See Table 7-4 for settings.
5	—	Reserved, should be cleared
6	ECLK	$clk\_out$ output enable. Used in conjunction with HID0[SBCLK] and the $\overline{hreset}$ signal to configure $clk\_out$ . See Table 7-4 for settings.
7	PAR	Disable precharge of $\overline{artry\_out}$ 0 Precharge of $\overline{artry\_out}$ enabled 1 Alters bus protocol slightly by preventing the processor from driving $\overline{artry\_out}$ to high (negated) state. If this is done, the integrated device must restore the signals to the high state.
8	DOZE	Doze mode enable. Operates in conjunction with MSR[POW]. 0 Doze mode disabled 1 Doze mode enabled. Doze mode is invoked by setting MSR[POW] while this bit is set. In doze mode, the PLL, time base, and snooping remain active.
9	NAP	Nap mode enable. Operates in conjunction with MSR[POW]. 0 Nap mode disabled 1 Nap mode enabled. Nap mode is invoked by setting MSR[POW] while this bit is set. In nap mode, the PLL and time base remain active.
10	SLEEP	Sleep mode enable. Operates in conjunction with MSR[POW]. 0 Sleep mode disabled 1 Sleep mode enabled. Sleep mode is invoked by setting MSR[POW] while this bit is set. $\overline{qreq}$ is asserted to indicate that the processor is ready to enter sleep mode. If the system logic determines that the processor may enter sleep mode, the quiesce acknowledge signal, $\overline{qack}$ , is asserted back to the processor. Once $\overline{qack}$ assertion is detected, the processor enters sleep mode after several processor clocks. At this point, the system logic may turn off the PLL by first configuring $pll\_cfg[0:6]$ to PLL bypass mode, then disabling $sysclk$ .

Table 7-3. e300 HID0 Bit Descriptions (continued)

Bits	Name	Function
11	DPM	Dynamic power management enable 0 Dynamic power management is disabled 1 Functional units enter a low-power mode automatically if the unit is idle. This does not affect operational performance and is transparent to software or any external hardware.
12–15	—	Reserved, should be cleared.
16	ICE	Instruction cache enable 0 The instruction cache is neither accessed nor updated. All pages are accessed as if they were marked cache-inhibited (WIM = x1x). Potential cache accesses from the bus (snoop and cache operations) are ignored. In the disabled state for the L1 caches, the cache tag state bits are ignored and all instruction fetches are propagated to the coherent system bus (CSB) as single-beat transactions. For those transactions, however, $\overline{ci}$ reflects the state of the I bit in the MMU for that page regardless of cache disabled status. ICE is zero at power-up. 1 The instruction cache is enabled
17	DCE	Data cache enable 0 The data cache is neither accessed nor updated. All pages are accessed as if they were marked cache-inhibited (WIM = x1x). Potential cache accesses from the bus (snoop and cache operations) are ignored. In the disabled state for the L1 caches, the cache tag state bits are ignored and all data read and write accesses are propagated to the CSB as single-beat transactions. For those transactions, however, $\overline{ci}$ reflects the state of the I bit in the MMU for that page regardless of cache disabled status. DCE is zero at power-up. 1 The data cache is enabled
18	ILOCK	Instruction cache lock 0 Normal operation 1 The entire instruction cache is locked (that is, all eight ways of the cache are locked). A locked cache supplies data normally on a hit, but the access is treated as a cache-inhibited transaction on a miss. On a miss, the transaction to the bus is single-beat; however, $\overline{ci}$ still reflects the state of the I bit in the MMU for that page independent of cache locked or disabled status. To prevent locking during a cache access, an <b>isync</b> instruction must precede the setting of ILOCK.
19	DLOCK	Data cache lock 0 Normal operation 1 The entire data cache is locked (that is, all eight ways of the cache are locked). A locked cache supplies data normally on a hit, but is treated as a cache-inhibited transaction on a miss. On a miss, the transaction to the bus is single-beat; however, $\overline{ci}$ still reflects the state of the I bit in the MMU for that page independent of cache locked or disabled status. A snoop hit to a locked L1 data cache performs as if the cache were not locked. A cache block invalidated by a snoop remains invalid until the cache is unlocked. To prevent locking during a cache access, a <b>sync</b> instruction must precede the setting of DLOCK.
20	ICFI	Instruction cache Flash invalidate 0 The instruction cache is not invalidated. The bit is cleared when the invalidation operation begins (usually the next cycle after the write operation to the register). The instruction cache must be enabled for the invalidation to occur. 1 An invalidate operation is issued that marks the state of each instruction cache block as invalid. Cache access is blocked during this time. Setting ICFI clears all the valid bits of the blocks and the PLRU bits to point to way L0 of each set. For the e300 core, the proper use of the ICFI and DCFI bits is to set and clear them with two consecutive <b>mtspr</b> operations.

Table 7-3. e300 HID0 Bit Descriptions (continued)

Bits	Name	Function
21	DCFI	Data cache Flash invalidate 0 The data cache is not invalidated. The bit is cleared when the invalidation operation begins (usually the next cycle after the write operation to the register). The data cache must be enabled for the invalidation to occur. 1 An invalidate operation is issued that marks the state of each data cache block as invalid without writing back modified cache blocks to memory. Cache access is blocked during this time. Bus accesses to the cache are signaled as a miss during invalidate-all operations. Setting DCFI clears all the valid bits of the blocks and the PLRU bits to point to way L0 of each set. For the e300 core, the proper use of the ICFI and DCFI bits is to set and clear them with two consecutive <b>mtspr</b> operations.
22–23	—	Reserved, should be cleared.
24	IFEM	Enable M bit on bus for instruction fetches 0 M bit not reflected on bus for instruction fetches. Instruction fetches are treated as nonglobal on the bus. 1 Instruction fetches reflect the M bit from the WIM settings
25	DECAREN	Decrementer auto reload 0 Normal operation. 1 Decrementer loads last mtdec value for precise periodic interrupt.
26	—	Reserved, should be cleared.
27	FBIOB	Force branch indirect on the bus 0 Register indirect branch targets are fetched normally 1 Forces register indirect branch targets to be fetched externally
28	ABE	Address broadcast enable. Controls whether certain address-only operations (such as cache operations) are broadcast on the bus. 0 Address-only operations affect only local caches and are not broadcast 1 Address-only operations are broadcast on the bus Affected instructions are <b>dcbi</b> , <b>dcbf</b> , and <b>dcbst</b> . Note that these cache control instruction broadcasts are not snooped by the e300 core. Refer to Section 4.3.3, “Data Cache Control,” for more information.
29–30	—	Reserved, should be cleared.
31	NOOPTI	No-op the data cache touch instructions 0 The <b>dcbt</b> and <b>dcbst</b> instructions are enabled 1 The <b>dcbt</b> and <b>dcbst</b> instructions are no-oped internal to the e300 core

Table 7-4 shows how HID0[ECLK] and HID0[SBCLK] are used to configure the *clk\_out* signal.

Table 7-4. Using HID0[ECLK] and HID0[SBCLK] to Configure *clk\_out*

$\overline{hreset}$	ECLK	SBCLK	<i>clk_out</i>
Asserted	x	x	Bus clock (small pulse for every rising edge of sysclk)
Negated	0	0	Clock output off
	0	1	Core clock/2
	1	0	Core clock
	1	1	Bus clock



Table 7-5 shows the bit definitions for HID1

**Table 7-5. HID1 Bit Descriptions**

Bits	Name	Description
0	PC0	PLL configuration bit 0 (read-only)
1	PC1	PLL configuration bit 1 (read-only)
2	PC2	PLL configuration bit 2 (read-only)
3	PC3	PLL configuration bit 3 (read-only)
4	PC4	PLL configuration bit 4 (read-only)
5	PC5	PLL configuration bit 5 (read-only)
6	PC6	PLL configuration bit 6 (read-only)
7–31	—	Reserved, should be cleared

**Note:** The clock configuration bits reflect the state of the *pll\_cfg[0:6]* signals.

Table 7-6 shows the bit definitions for HID2.

**Table 7-6. e300HID2 Bit Descriptions**

Bits	Name	Description
0–3	—	Reserved, should be cleared.
4	LET	True little-endian. This bit enables true little-endian mode operation for instruction and data accesses. This bit is set to reflect the state of the <i>tle</i> signal at the negation of <i>hreset</i> . This bit is used in conjunction with MSR[LE] to determine the endian mode of operation. 0 No function 1 True little-endian mode, when MSR[LE] = 1 Changing the value of this bit during normal operation is not recommended
5	IFEB	Instruction fetch burst extension. This bit enables the instruction fetch burst extension. 0 Instruction fetch burst extension disabled 1 Instruction fetch burst extension enabled
6	—	Reserved, should be cleared.
7	MESISTATE	MESI state enable. This bit enables the four-state MESI cache coherency protocol. 0 MESI disabled. The data cache uses a three-state MEI coherency protocol. 1 MESI enabled. The data cache uses a four-state MESI protocol.
8	IFEC	Instruction fetch cancel extension. This bit enables the instruction fetch cancel extension. 0 Instruction fetch cancel extension disabled 1 Instruction fetch cancel extension enabled
9	EBQS	Enable BIU queue sharing. This bit enables data cache queue sharing. 0 Data cache queue sharing disabled 1 Data cache queue sharing enabled
10	EBPX	Enable BIU pipeline extension. This bit enables the bus interface unit pipeline extension. 0 BIU pipeline extension disabled; 1 level pipeline 1 BIU pipeline extension enabled; 1-1/2 level pipeline

Table 7-6. e300HID2 Bit Descriptions (continued)

Bits	Name	Description
11	ELRW	Enable weighted LRU. This bit enables the use of an adjusted (weighted) LRU. 0 Normal operation. 1 The dcbt, dcbtst, and dcbz instructions use an adjusted (weighted) LRU such that they always select and replace the lowest unlocked way in the data cache.
12	NOKS	No kill for snoop. This bit enables the forcing of kill-type snoops to flush data instead of killing it. 0 Normal operation. 1 Forces write-with-kill snoops to flush instead of kill (snoop can never kill data).
13	HBE	High BAT enable. Regardless of the setting of HID2[HBE], these BATs are accessible by <b>mfspr</b> and <b>mtspr</b> . 0 IBAT[4–7] and DBAT[4–7] are disabled 1 IBAT[4–7] and DBAT[4–7] are enabled
14–15	—	Reserved, should be cleared.
16–18	IWLCK[0–2]	Instruction cache way-lock. Useful for locking blocks of instructions into the instruction cache for time-critical applications that require deterministic behavior. 000 no ways locked 001 way 0 locked 010 way 0 through way 1 locked 011 way 0 through way 2 locked 100 way 0 through way 2 locked in e300c3. 101 way 0 through way 2 locked in e300c3. 110 way 0 through way 2 locked in e300c3. 111 way 0 through way 2 locked in e300c3. Setting HID0[ILOCK] locks all ways.
19	ICWP	Instruction cache way protection. Used to protect locked ways in the instruction cache from being invalidated. 0 Instruction cache way protection disabled 1 Instruction cache way protection enabled
20–23	—	Reserved, should be cleared.
24–26	DWLCK[0–2]	Data cache way-lock. Useful for locking blocks of data into the data cache for time-critical applications where deterministic behavior is required. 000 no ways locked 001 way 0 locked 010 way 0 through way 1 locked 011 way 0 through way 2 locked 100 way 0 through way 2 locked in e300c3. 101 way 0 through way 2 locked in e300c3. 110 way 0 through way 2 locked in e300c3. 111 way 0 through way 2 locked in e300c3. Setting HID0[DLOCK] locks all ways.
27–31	—	Reserved, should be cleared.

## 7.4.2 Instruction Set and Addressing Modes

The following sections describe the PowerPC instruction set and addressing modes in general.

### 7.4.2.1 PowerPC Instruction Set and Addressing Modes

All PowerPC instructions are encoded as single-word (32-bit) opcodes. Instruction formats are consistent among all instruction types, permitting efficient decoding to occur in parallel with operand accesses. This fixed instruction length and consistent format simplifies instruction pipelining.

The PowerPC instructions are divided into the following categories:

- Integer instructions (includes computational and logical instructions)
  - Integer arithmetic instructions
  - Integer compare instructions
  - Integer logical instructions
  - Integer rotate and shift instructions
- Floating-point instructions (includes floating-point computational instructions and instructions that affect the FPSCR)
  - Floating-point arithmetic instructions
  - Floating-point multiply/add instructions
  - Floating-point rounding and conversion instructions
  - Floating-point compare instructions
  - Floating-point status and control instructions
- Load/store instructions (includes integer and floating-point load and store instructions)
  - Integer load and store instructions
  - Integer load and store multiple instructions
  - Floating-point load and store
  - Primitives used to construct atomic memory operations (**lwarx** and **stwex**. instructions)
- Flow control instructions (includes branching instructions, condition register logical instructions, trap instructions, and other instructions that affect the instruction flow)
  - Branch and trap instructions
  - Condition register logical instructions
- Processor control instructions (includes instructions used for synchronizing memory accesses and managing caches, TLBs, and the segment registers)
  - Move to/from SPR instructions
  - Move to/from MSR
  - Move to/from PMR
  - Synchronize
  - Instruction synchronize
- Memory control instructions (includes instructions that provide control of caches, TLBs, and segment registers)
  - Supervisor-level cache management instructions
  - Translation lookaside buffer management instructions. Note that there are additional implementation-specific instructions.

- User-level cache instructions
- Segment register manipulation instructions

The e300 core implements the following instructions defined as optional by the PowerPC architecture:

- Floating Select (**fsel**)
- Floating Reciprocal Estimate Single-Precision (**fres**)
- Floating Reciprocal Square Root Estimate (**frsqrte**)
- Store Floating-Point as Integer Word (**stfiwx**)

Note that this grouping of instructions does not indicate the execution unit that executes a particular instruction or group of instructions.

Integer instructions operate on byte, half-word, and word operands. Floating-point instructions operate on single-precision (one word) and double-precision (one double word) floating-point operands. The PowerPC architecture uses instructions that are 4-bytes long and word-aligned. It provides for byte, half-word, and word operand loads and stores between memory and a set of 32 GPRs. It also provides for word and double-word operand loads and stores between memory and a set of 32 FPRs.

Computational instructions do not modify memory. To use a memory operand in a computation and then modify the same or another memory location, the memory contents must be loaded into a register, modified, and then written back to the target location with distinct instructions.

The core follows the program flow when it is in the normal execution state. However, the flow of instructions can be interrupted directly by the execution of an instruction or by an asynchronous event. Either kind of interrupt may cause one of several components of the system software to be invoked.

#### 7.4.2.2 Implementation-Specific Instruction Set

The e300 core instruction set is defined as follows:

- The core provides hardware support for all 32-bit PowerPC instructions.
- The core provides two implementation-specific instructions used for software table search operations following TLB misses:
  - Load Data TLB Entry (**tlbld**)
  - Load Instruction TLB Entry (**tlbli**)
- The core implements the following instruction that is added to support critical interrupts (also supported on the G2\_LE). This is a supervisor-level, context synchronizing instruction.
  - Return from Critical Interrupt (**rftci**)
- The core implements the following instruction that is added to support easy start-up initialization or reloading of the instruction cache.
  - Instruction Cache Block Touch (**icbt**)
- The core provides the following performance monitor instructions:
  - Move to Performance Monitor Register (**mtpmr**)
  - Move from Performance Monitor Register (**mfpmr**)

## 7.4.3 Cache Implementation

The following sections describe the general cache characteristics as implemented in the PowerPC architecture and the core implementation.

### 7.4.3.1 PowerPC Cache Characteristics

The PowerPC architecture does not define hardware aspects of cache implementations. The e300 core controls the following memory access modes on a page or block basis:

- Write-back/write-through mode
- Caching-inhibited mode
- Memory coherency

Note that in the core, a cache block is defined as eight words. The VEA defines cache management instructions that provide a means by which the application programmer can affect the cache contents.

### 7.4.3.2 Implementation-Specific Cache Organization

The e300c3 provides 16-Kbyte, four-way set-associative instruction and data caches. The caches are physically addressed, and the data cache can operate in either write-back or write-through mode as specified by the PowerPC architecture.

The data cache is configured as 128 sets of 4 blocks each on the e300c3. Each block consists of 32 bytes, 2 state bits, and an address tag. The two state bits implement the three-state MEI (modified/exclusive/invalid) protocol. Each block contains eight 32-bit words. Note that the PowerPC architecture defines the term ‘block’ as the cacheable unit. For the core, the block size is equivalent to a cache line. A block diagram of the data cache organization is shown in [Figure 7-3](#).

The instruction cache is configured as 128 sets of 4 blocks each on the e300c3. Each block consists of 32 bytes, an address tag, and a valid bit. The instruction cache may not be written to, except through a block fill operation. In the e300 core, the instruction cache is blocked only until the critical load completes. The e300 core supports instruction fetching from other instruction cache lines following the forwarding of the critical-first-double-word of a cache line load operation. Successive instruction fetches from the cache line being loaded are forwarded, and accesses to other instruction cache lines can proceed during the cache line load operation. The instruction cache is not snooped, and cache coherency must be maintained by software. A fast hardware invalidation capability is provided to support cache maintenance.

The e300c3 data cache is configured as 128 sets of four blocks per set. The organization of the data cache is shown in Figure 7-3.

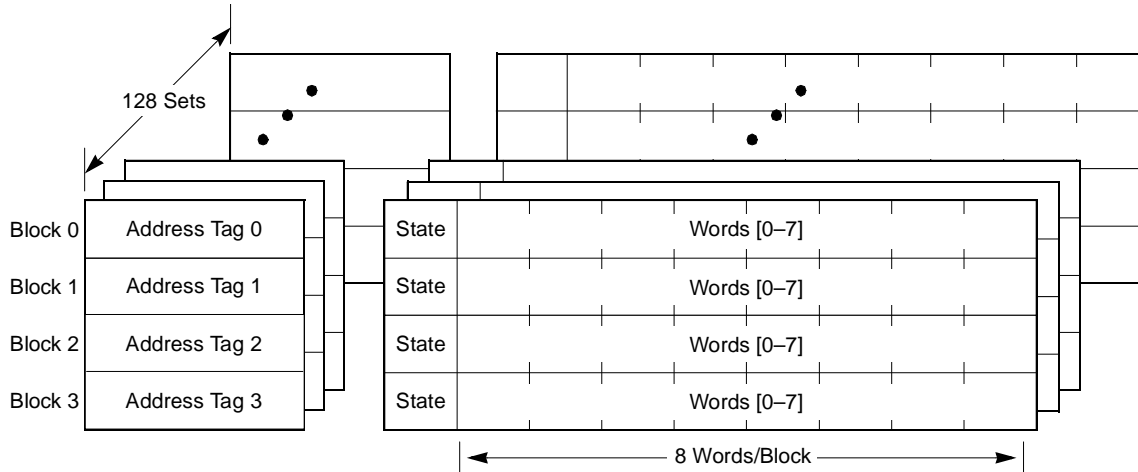


Figure 7-3. e300c3 Data Cache Organization

Each cache block contains eight contiguous words from memory that are loaded from an 8-word boundary (that is, bits A[27–31] of the effective addresses are zero); thus, a cache block never crosses a page boundary. Misaligned accesses across a page boundary can incur a performance penalty.

The e300 core cache blocks are loaded in four beats of 64 bits each on the 64-bit data bus. The burst load is performed as critical-double-word-first. The data cache is blocked to internal accesses until the load completes; the instruction cache allows sequential fetching during a cache block load. In the core, the critical-double-word is simultaneously written to the cache and forwarded to the requesting unit, thus minimizing stalls due to load delays.

To ensure coherency among caches in a multiprocessor (or multiple caching-device) implementation, the core implements the MEI protocol during normal operation of the data cache. The new data cache MESI extension supports the additional fourth cache coherency shared state for the data cache. To support this feature, the shared signal, *shd*, has been added to the bus interface. Although the MESI protocol is supported by the e300 core, it is not implemented on MPC8308. The following four states indicate the state of the cache block:

- **Modified**—The cache block is modified with respect to system memory; that is, data for this address is valid only in the cache and not in system memory.
- **Exclusive**—This cache block holds valid data that is identical to the data at this address in system memory. No other cache has this data.
- **Shared**—Only available if HID2[MESISTATE] register bit is set. The address block is valid in the cache and in at least one other cache. This block is always consistent with system memory. That is, the shared state is shared-unmodified; there is no shared-modified state. Although the MESI protocol is supported by the e300 core, it is not implemented on MPC8308.
- **Invalid**—This cache block does not hold valid data.

Cache coherency is enforced by on-chip bus snooping logic. Because the e300 core data cache tags are single-ported, a simultaneous load/store and snoop access represents a resource contention. The snoop access is given first access to the tags. The load or store then occurs on the clock following the snoop.

Parity is now integrated into both instruction and data cache memory. A machine check interrupt is now taken upon the detection of an instruction or data cache parity error. Parity is checked whenever valid data is returned from the instruction or data cache for a cache hit or whenever valid data is read out of the cache for a castout or snoop-push operation.

### 7.4.3.3 Instruction and Data Cache Way-Locking

The e300 core implements instruction and data cache way-locking, which guarantees that certain memory accesses hit in the cache. This provides deterministic access times for those accesses.

## 7.4.4 Interrupt Model

This section describes the PowerPC interrupt model and the e300 core implementation specifically.

### 7.4.4.1 PowerPC Interrupt Model

The PowerPC interrupt mechanism allows the core to change to supervisor state as a result of external signals, errors, or unusual conditions arising in the execution of instructions. The conditions that can cause interrupts are called exceptions. When interrupts occur, information about the state of the core is saved to certain registers and the core begins execution at an address (interrupt vector) predetermined for each interrupt type. Interrupts are processed in supervisor mode.

Some interrupts, such as program interrupts, can be triggered by a broad range of exception conditions. Other interrupts, such as the decremter interrupt, have only a single exception condition. Although multiple exception conditions can map to a single interrupt vector, a more specific condition may be determined by examining a register associated with the interrupt—for example, the DSISR and the FPSCR. Additionally, some exception conditions can be explicitly enabled or disabled by software.

The PowerPC architecture requires that interrupts be handled in program order; therefore, although a particular implementation may recognize exception conditions out of order, they are presented strictly in order. When an instruction-caused interrupt is recognized, any unexecuted instructions that appear earlier in the instruction stream, including any that have not yet entered the execute stage, are required to complete before the interrupt is taken. Any interrupts caused by those instructions are handled first. Likewise, asynchronous, precise interrupts are recognized when they occur, but are not handled until the instruction currently in the completion stage successfully completes execution or generates an interrupt, and the completed store queue is emptied.

Unless a catastrophic condition causes a system reset or machine check interrupt, only one interrupt is handled at a time. If, for example, a single instruction encounters multiple interrupt conditions, those conditions are handled sequentially. After the interrupt handler completes, the instruction execution continues until the next interrupt condition is encountered. However, in many cases there is no attempt to re-execute the instruction. This method of recognizing and handling interrupts sequentially guarantees that interrupts are recoverable.

To prevent the program state from being lost due to a system reset, a machine check interrupt or an instruction-caused interrupt in the interrupt handler should save the information stored in SRR0 and SRR1 early and before enabling external interrupts.

The PowerPC architecture supports four types of interrupts:

- Synchronous, precise

These are caused by instructions. All instruction-caused interrupts are handled precisely; that is, the machine state at the time the interrupt occurs is known and can be completely restored. This means that (excluding the trap and system call interrupts) the address of the faulting instruction is provided to the interrupt handler and neither the faulting instruction nor subsequent instructions in the code stream completes execution before the interrupt is taken. Once the interrupt is processed, execution resumes at the address of the faulting instruction (or at an alternate address provided by the interrupt handler). When an interrupt is taken due to a trap or system call instruction, execution resumes at an address provided by the handler.

- Synchronous, imprecise

The PowerPC architecture defines two imprecise floating-point exception modes: recoverable and nonrecoverable. Even though the core provides a means to enable the imprecise modes, it implements these modes identically to the precise mode (that is, all enabled floating-point exceptions are always precise on the core).

- Asynchronous, maskable

The external system management interrupt (SMI) and decremter interrupts are maskable, asynchronous interrupts. When these interrupts occur, their handling is postponed until the next instruction and any of its associated interrupts complete execution. If there are no instructions in the execution units, the interrupt is taken immediately upon determination of the correct restart address (for loading SRR0).

- Asynchronous, nonmaskable

The system reset and the machine check interrupt are nonmaskable, asynchronous interrupts. They may not be recoverable, or they may provide a limited degree of recoverability. All interrupts report recoverability through MSR[RI].

#### 7.4.4.2 Implementation-Specific Interrupt Model

As specified by the PowerPC architecture, all interrupts can be described as either precise or imprecise and either synchronous or asynchronous. Asynchronous interrupts (some of which are maskable) are caused by events external to the processor's execution; synchronous interrupts, which are all handled precisely by



the e300 core, are caused by instructions. A system management interrupt is an implementation-specific interrupt. The interrupt classes are shown in [Table 7-7](#).

**Table 7-7. Interrupt Classifications**

Synchronous/Asynchronous	Precise/Imprecise	Interrupt Type
Asynchronous, nonmaskable	Imprecise	Machine check System reset
Asynchronous, maskable	Precise	External interrupt Decrementer System management interrupt Critical interrupt
Synchronous	Precise	Instruction-caused interrupts

Although interrupts have other characteristics, such as whether they are maskable, the distinctions shown in [Table 7-7](#) define categories of interrupts that the core handles uniquely. Note that [Table 7-7](#) includes no synchronous, imprecise instructions. While the PowerPC architecture supports imprecise handling of floating-point exceptions, the core implements floating-point exception modes as precise.

The e300 core interrupts and exception conditions that cause them are listed in [Table 7-8](#).

**Table 7-8. Exceptions and Interrupts**

Interrupt Type	Vector Offset (hex)	Exception Conditions
Reserved	00000	—
System reset	00100	Caused by the assertion of either $\overline{hreset}$ .
Machine check	00200	Caused by the assertion of the $\overline{tea}$ signal during a data bus transaction, assertion of $\overline{mcp}$ , an address or data parity error, or an instruction or data cache parity error. Note that the e300 has SRR1 register values that are different from the G2/G2_LE cores' when a machine check occurs.
DSI	00300	Determined by the bit settings in the DSISR, listed as follows: <ul style="list-style-type: none"> <li>1 Set if the translation of an attempted access is not found in the primary hash table entry group (HTEG), or in the rehashed secondary HTEG, or in the range of a DBAT register; otherwise cleared</li> <li>4 Set if a memory access is not permitted by the page or DBAT protection mechanism; otherwise cleared</li> <li>6 Set for a store operation and cleared for a load operation</li> <li>9 Set if a data address breakpoint interrupt occurs when the data [0–28] in the DABR or DABR2 matches the next data access (load or store instruction) to complete in the completion unit. The different breakpoints are enabled as follows: <ul style="list-style-type: none"> <li>• Write breakpoints enabled when DABR[30] is set</li> <li>• Read breakpoints enabled when DABR[31] is set</li> </ul> </li> </ul>
ISI	00400	Caused when an instruction fetch cannot be performed for any of the following reasons: <ul style="list-style-type: none"> <li>• The effective (logical) address cannot be translated. That is, there is a page fault for this portion of the translation, so an ISI interrupt must be taken to load the PTE (and possibly the page) into memory.</li> <li>• The fetch access violates memory protection (indicated by SRR1[4] set). If the key bits (Ks and Kp) in the segment register and the PP bits in the PTE are set to prohibit read access, instructions cannot be fetched from this location.</li> </ul>

Table 7-8. Exceptions and Interrupts (continued)

Interrupt Type	Vector Offset (hex)	Exception Conditions
External interrupt	00500	Caused when MSR[EE] = 1 and the $\overline{int}$ signal is asserted.
Alignment	00600	Caused when the core cannot perform a memory access for any of the reasons described below: <ul style="list-style-type: none"> <li>The operand of a floating-point load or store instruction is not word-aligned.</li> <li>The operands of <b>lmw</b>, <b>stmw</b>, <b>lwarx</b>, and <b>stwcx</b>. instructions are not aligned.</li> <li>The instruction is <b>lswi</b>, <b>lswx</b>, <b>stswi</b>, <b>stswx</b>, and the core is in little-endian mode. Note that PowerPC little-endian mode is not supported on the e300 core.</li> <li>The operand of <b>dcbz</b> is in memory that is write-through-required or caching-inhibited.</li> </ul>
Program	00700	Caused by one of the following exception conditions, which correspond to bit settings in SRR1 and arise during execution of an instruction. Floating-point enabled exception—A floating-point enabled exception condition is generated when the following condition is met: (MSR[FE0]   MSR[FE1]) and FPSCR[FEX] is 1. <ul style="list-style-type: none"> <li>FPSCR[FEX] is set by the execution of a floating-point instruction that causes an enabled exception or by the execution of one of the Move to FPSCR instructions that results in both an exception condition bit and its corresponding enable bit being set in the FPSCR.</li> <li>Illegal instruction—An illegal instruction program interrupt is generated when execution of an instruction is attempted with an illegal opcode or illegal combination of opcode and extended opcode fields (including PowerPC instructions not implemented in the core), or when execution of an optional instruction not provided in the core is attempted (these do not include those optional instructions that are treated as no-ops).</li> <li>Privileged instruction—A privileged instruction program interrupt is generated when the execution of a privileged instruction is attempted and the MSR register user privilege bit, MSR[PR], is set. In the e300 core, this interrupt is generated for <b>mtspr</b> or <b>mfspr</b> with an invalid SPR field if SPR[0] = 1 and MSR[PR] = 1. This may not be true for all cores that implement the PowerPC architecture.</li> <li>Trap—A trap type program interrupt is generated when any of the conditions specified in a trap instruction are met.</li> </ul>
Floating-point unavailable	00800	Caused by an attempt to execute a floating-point instruction (including floating-point load, store, and move instructions) when the floating-point available bit (MSR[FP]) is cleared.
Decrementer	00900	Occurs when DEC[0] changes from 0 to 1. This interrupt is enabled with MSR[EE].
Critical interrupt	00A00	Taken when $\overline{cint}$ is asserted and MSR[CE] = 1.
Reserved	00B00–00BFF	—
System call	00C00	Occurs when a System Call ( <b>sc</b> ) instruction is executed.
Trace	00D00	Taken when MSR[SE] = 1 or when the currently completing instruction is a branch and MSR[BE] = 1.
Reserved	00E00	The e300 core does not generate an interrupt to this vector. Other devices may use this vector for floating-point assist interrupts.
Performance monitor	00F00	Caused when a configured PM counter using the pm_event_in to transition overflows.
Instruction translation miss	01000	Caused when the effective address for an instruction fetch cannot be translated by the ITLB.
Data load translation miss	01100	Caused when the effective address for a data load operation cannot be translated by the DTLB.

Table 7-8. Exceptions and Interrupts (continued)

Interrupt Type	Vector Offset (hex)	Exception Conditions
Data store translation miss	01200	Caused when the effective address for a data store operation cannot be translated by the DTLB, or when a DTLB hit occurs and the change bit in the PTE must be set due to a data store operation.
Instruction address breakpoint	01300	Occurs when the address (bits 0–29) in the IABR matches the next instruction to complete in the completion unit, and IABR[30] is set. Note that the e300 core also implements IABR2, which functions identically to IABR.
System management interrupt	01400	Caused when MSR[EE] = 1 and the $\overline{smi}$ input signal is asserted.
Reserved	01500–02FFF	—

## 7.4.5 Memory Management

The following sections describe the memory management features of the PowerPC architecture and the e300 core implementation, respectively.

### 7.4.5.1 PowerPC Memory Management

The primary functions of the MMU are to translate logical (effective) addresses to physical addresses for memory accesses and to provide access protection on blocks and pages of memory.

The core generates two types of accesses that require address translation: instruction accesses and data accesses to memory generated by load and store instructions.

The PowerPC MMU and interrupt model support demand-paged virtual memory. Virtual memory management permits execution of programs larger than the size of physical memory; demand-paged implies that individual pages are loaded into physical memory from system memory only when they are first accessed by an executing program.

The hashed page table is a variable-sized data structure that defines the mapping between virtual page numbers and physical page numbers. The page table size is a power of two, and its starting address is a multiple of its size.

The page table contains a number of page-table entry groups (PTEGs). A PTEG contains eight page-table entries (PTEs) of 8 bytes each; therefore, each PTEG is 64 bytes long. PTEG addresses are entry points for table search operations.

Address translations are enabled by setting bits in the MSR. MSR[IR] enables instruction address translations, and MSR[DR] enables data address translations.

### 7.4.5.2 Implementation-Specific Memory Management

The instruction and data memory management units in the e300 core provide 4 Gbytes of logical address space accessible to supervisor and user programs with a 4-Kbyte page size and 256-Mbyte segment size. Block sizes range from 128 Kbytes to 256 Mbytes and are software selectable. In addition, the core uses

an interim 52-bit virtual address and hashed page tables for generating 32-bit physical addresses. The MMUs in the e300 core rely on the interrupt processing mechanism for the implementation of the paged virtual memory environment and for enforcing protection of designated memory areas.

Instruction and data TLBs provide address translation in parallel with the on-chip cache access, incurring no additional time penalty in the event of a TLB hit. A TLB is a cache of the most recently used page table entries. Software is responsible for maintaining the consistency of the TLB with memory. The core TLBs are 64-entry, two-way, set-associative caches that contain instruction and data address translations. The core provides hardware assist for software table search operations through the hashed page table on TLB misses. Supervisor software can invalidate TLB entries selectively.

For instructions and data that correspond to block address translation, the e300 core provides independent eight-entry BAT arrays. These entries define blocks that can vary from 128 Kbytes to 256 Mbytes. The BAT arrays are maintained by system software. HID2[HBE] is added to the e300 for enabling or disabling the four additional pairs of BAT registers. However, regardless of the setting of HID2[HBE], these BATs are accessible by **mfspr** and **mtspr**.

As specified by the PowerPC architecture, the hashed page table is a variable-sized data structure that defines the mapping between virtual page numbers and physical page numbers. The page table size is a power of two, and its starting address is a multiple of its size.

Also as specified by the PowerPC architecture, the page table contains a number of PTEGs. A PTEG contains 8 PTEs of 8 bytes each; therefore, each PTEG is 64 bytes long. PTEG addresses are entry points for table search operations.

## 7.4.6 Instruction Timing

The e300 core is a pipelined superscalar processor core. Because instruction processing is reduced into a series of stages, an instruction does not require all of the resources of an execution unit at the same time. For example, after an instruction completes the decode stage, it can pass on to the next stage, while the subsequent instruction can advance into the decode stage. This improves the throughput of the instruction flow. For example, it may take three cycles for a single floating-point instruction to execute, but if there are no stalls in the floating-point pipeline, a series of floating-point instructions can have a throughput of one instruction per cycle.

The core instruction pipeline has four major pipeline stages, described as follows:

- The fetch pipeline stage primarily involves retrieving instructions from the memory system and determining the location of the next instruction fetch. Additionally, if possible, the BPU decodes branches during the fetch stage and folds out branch instructions before the dispatch stage.
- The dispatch pipeline stage is responsible for decoding the instructions supplied by the instruction fetch stage and determining which of the instructions are eligible to be dispatched in the current cycle. In addition, the source operands of the instructions are read from the appropriate register file and dispatched with the instruction to the execute pipeline stage. At the end of the dispatch pipeline stage, the dispatched instructions and their operands are latched by the appropriate execution unit.
- In the execute pipeline stage, each execution unit with an instruction executes the selected instruction (perhaps over multiple cycles), writes the instruction's result into the appropriate rename register, and notifies the completion stage when the execution has finished. In the case of

an internal interrupt, the execution unit reports the interrupt to the completion/write-back pipeline stage and discontinues instruction execution until the interrupt is handled. The interrupt is not signaled until that instruction is the next to be completed. Execution of most floating-point instructions is pipelined within the FPU, allowing up to three instructions to execute in the FPU concurrently. The FPU pipeline stages are multiply, add, and round-convert. The LSU has two pipeline stages: the first stage, for effective address calculation and MMU translation and the second for accessing data in the cache.

- The complete/write-back pipeline stage maintains the correct architectural machine state and transfers the contents of the rename registers to the GPRs and FPRs as instructions are retired. If the completion logic detects an instruction causing an interrupt, all subsequent instructions are canceled, their execution results in rename registers are discarded, and instructions are fetched from the correct instruction stream.

A superscalar processor core issues multiple, independent instructions into multiple pipelines, allowing instructions to execute in parallel. The e300c1 core has independent execution units for: integer instructions, floating-point instructions, branch instructions, load/store instructions, and system register instructions. The e300c3 provides two IUs, which improves the throughput of integer instructions. The e300c3 provides two integer units for greater integer instruction throughput along with enhanced multipliers in each IU that reduce the multiply instruction latency to a maximum of two cycles. The IU and the FPU each have dedicated register files for maintaining operands (GPRs and FPRs, respectively), allowing integer and floating-point calculations to occur simultaneously without interference.

The core provides support for single-cycle store, and it provides an adder/comparator in the system register unit that allows the dispatch and execution of multiple integer add and compare instructions on each cycle.

Because the PowerPC architecture can be applied to such a wide variety of implementations, instruction timing among processor cores varies accordingly.

### 7.4.7 Core Interface

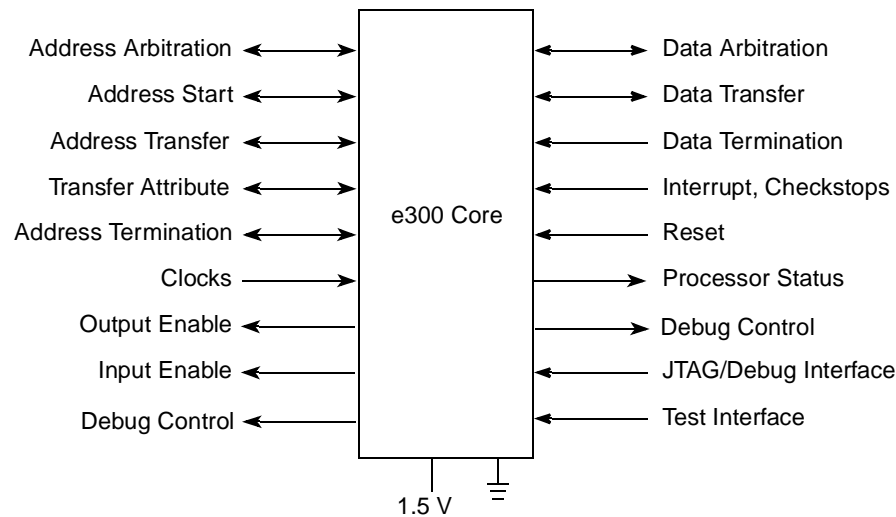
The core interface is specific for each processor core implementation.

The MPC8308 contains an internal coherent system bus (CSB) that interfaces the processor core to the peripheral logic. In the case of the MPC8308, the CSB system logic decodes e300-initiated transactions and directs all accesses to the appropriate interface.

The e300 core can operate at a variety of frequencies allowing the designer to trade off performance for power consumption. The processor core is clocked from a separate PLL, which is referenced to the CSB frequency. This allows the processor core and the peripheral logic to operate at different frequencies.

The e300 core provides a versatile core interface that allows for a wide range of implementations. The interface includes a 32-bit address bus, a 64-bit data bus, and 56 control and information signals (see [Figure 7-4](#)). The core interface allows for address-only transactions, as well as address and data transactions. The core control and information signals include the address arbitration, address start,

address transfer, transfer attribute, address termination, data arbitration, data transfer, data termination, and core state signals. Test and control signals provide diagnostics for selected internal circuits.



**Figure 7-4. Core Interface**

The core interface supports bus pipelining, allowing the address tenure of one transaction to overlap the data tenure of another. The extent of the pipelining depends on external arbitration and control circuitry. Similarly, the core supports split-bus transactions for systems with multiple potential bus masters; one device can have mastership of the address bus while another has mastership of the data bus. Allowing multiple bus transactions to occur simultaneously increases the available bus bandwidth for other activity and, as a result, improves performance.

The core clocking structure allows the bus to operate at integer multiples of the core cycle time.

The following sections describe the core bus support for memory operations. Note that some signals perform different functions depending on the addressing protocol used.

#### 7.4.7.1 Memory Accesses

The e300 core CSB is a 64-bit data bus.

With a 64-bit CSB, memory accesses allow transfer sizes of 8, 16, 24, 32, 40, 48, 56, or 64 bits in one bus clock cycle. Data transfers occur in either single-beat transactions or four-beat burst transactions. Single-beat transactions are caused by noncached accesses that access memory directly (that is, reads and writes when caching is disabled, caching-inhibited accesses, and stores in write-through mode). Four-beat burst transactions, which always transfer an entire cache block (32 bytes), are initiated when a line is read from or written to memory.

#### 7.4.7.2 Signals

The e300 core signals are grouped as follows:

- Interrupts/Resets

These signals include the external interrupt signal ( $\overline{int}$ ), critical interrupt signal ( $\overline{cint}$ ), checkstop signals, performance monitor signal ( $pm\_event\_in$ ) via the PM counters, and both soft reset and hard reset signals. They are used to interrupt and, under various conditions, to reset the core.

- JTAG/debug interface signals

The JTAG (based on the IEEE 1149.1 standard) interface and debug unit provides a serial interface to the system for performing monitoring and boundary tests. Two additional signals are added to the e300 core to allow observation of the internal clock state of the core ( $\overline{stopped}$ ) and to allow the external input to force the core into a halted state ( $ext\_halt$ ).

- Core status and control

These signals include the memory reservation signal, machine quiesce control signals, time base/decrementer clock base enable signal, and the  $\overline{tlb\,isync}$  signal.

- Clock control

These signals provide for system clock input and frequency control.

- Test interface signals

Signals like address matching, combinational matching, and watchpoint are used in the core for production testing.

- Transfer attribute signals

These signals provide information about the type of transfer, such as the transfer size and whether the transaction is bursted, write-through, or cache-inhibited.

## 7.4.8 Debug Features

Some new debug features are specific to the e300 core. Accesses to the debug facilities are available only in supervisor mode by using the **mtspr** and **mf spr** instructions. The e300 provides the following additional feature in the JTAG/debug interface: Inclusion of breakpoint status and control pins:  $\overline{stopped}$  and  $ext\_halt$ .

### 7.4.8.1 Breakpoint Signaling

The breakpoint signaling provided on the e300 core allows observability of breakpoint matches external to the core. The  $\overline{iabr}$ ,  $\overline{iabr2}$ ,  $\overline{dabr}$ , and  $\overline{dabr2}$  breakpoint signals are asserted for at least one bus clock cycle when the respective breakpoint occurs. The status of the run state of the e300 core is indicated by the  $\overline{stopped}$  pin. An asynchronous external breakpoint can be asserted to the e300 core using the  $ext\_halt$  pin:

- When DBCR and IBCR are configured for an OR combinational signal type, the breakpoint signals  $\overline{iabr}$ ,  $\overline{iabr2}$  and  $\overline{dabr}$ ,  $\overline{dabr2}$  reflect their respective breakpoints.
- When the DBCR and IBCR are configured for AND combinational signal type, only the  $\overline{iabr2}$  and  $\overline{dabr2}$  breakpoint signals are asserted after the AND condition is met (that is, both instruction breakpoints occurred or both data breakpoints occurred).
- When the  $core\_stopped$  pin is asserted, the e300 core has entered a stopped state and all internal clocking has stopped, indicating that a hardware debug event has occurred.
- The  $ext\_halt$  input pin can be used to force the core into halted state. The halted state may be a hardstop, conditional upon the HARDSTOP condition being set through the JTAG/debug interface

## 7.5 Differences Between Cores

The e300 core has similar functionality to the G2\_LE core. [Table 7-9](#) describes the differences between the G2\_LE and the e300.

**Table 7-9. Differences Between e300 and G2\_LE Cores**

e300 Core	G2_LE Core	Impact
New HID0 bits	—	The e300 core has a new HID0 bit defined to enable cache parity error reporting (ECPE).
New HID1 bits	—	The e300 core has new HID1 bits defined to extend the number of PLL configuration signals to seven (PC5, PC6).
New HID2 bits	—	The e300 core has new HID2 bits defined to support instruction fetch bursting (IFEB), MESI coherency protocol (MESI), instruction fetch cancels (IFEC), data cache queue sharing (EBQS), pipelining extension (EBPX), additional cache way locking (IWLCK and DWLCK), and instruction cache way protection (ICWP).
New PVR register value	—	The processor version register values differ.
New IBCR and DBCR bits	—	The e300 core has new IBCR[IABRSTAT, IABR2STAT] and DBCR[DABR1STAT, DABR2STAT] fields to provide instruction and data address breakpoint status.
—	16-Kbyte, four-way, set-associative, instruction and data caches	Some e300 cores may have different cache sizes than the G2_LE. For detailed information, see <i>e300 PowerPC Core Reference Manual</i> .
L1 cache parity	—	The e300 core supports parity for both instruction and data caches; the G2_LE does not support cache parity.
MEI or MESI coherency protocols	MEI protocol only	The e300 supports two coherency protocols: MEI and MESI; the G2_LE only supports the MEI protocol.
Instruction cancel extension	—	The e300 instruction cancel mechanism improves utilization of instruction cache by supporting 'hits-under-cancels' and 'misses-under-cancels'; the G2_LE requires the cancel to complete before new instruction fetches can begin.
Instruction fetch bursts to caching-inhibited space	Single-beat instruction fetches to caching-inhibited space	The e300's instruction fetch burst extension allows all caching-inhibited instruction fetches to be performed on the bus as burst transactions, even though the instructions are not cached. This improves performance for instruction space that is caching-inhibited, because up to eight instructions are returned with one bus operation. The G2_LE core must use single-beat instruction fetches for caching-inhibited space, returning only two instructions per bus operation.
Instruction cache way protection	—	The e300 core can protect locked ways in the instruction cache from invalidation; the G2_LE does not support instruction cache way protection.



Table 7-9. Differences Between e300 and G2\_LE Cores (continued)

e300 Core	G2_LE Core	Impact
Data cache queue sharing	—	The e300 has a new data cache queue sharing extension that allows the two burst-write queues in the bus unit to be used interchangeably for cache replacements and snoop pushes. Thus, the data cache can support two outstanding cache replacements or two outstanding snoop push operations on the bus at any given time.
<b>icbt</b> instruction	—	The e300 supports a new instruction cache block touch instruction that facilitates preloading the instruction cache before locking; the G2_LE core requires speculatively fetching instructions before locking the instruction cache.
1-½-level bus pipelining	1-level bus pipelining	For the e300, a new transaction can complete an address tenure when the previous transaction has been granted the data bus; for the G2_LE, a new transaction must wait until the previous data tenure has completed before completing its address tenure.
PowerPC little-endian not supported	PowerPC little-endian supported	PowerPC little-endian is not supported in the e300 core, although true little-endian is fully supported.
Data retry mode removed	Data retry mode available	$\overline{drtry}$ and $drtrymode$ is no longer supported on the e300 and future versions.
External control instructions removed	External control instructions available	The <b>eciwx</b> and <b>ecowx</b> instruction pair is not supported on the e300 core. These are optional instructions in the PowerPC architecture.
Reduced pin mode removed	Reduced pin mode available	Reduced pinout mode and the signal <i>redpinmode</i> is not supported in the e300 core.



# Chapter 8

## Integrated Programmable Interrupt Controller (IPIC)

This chapter describes the integrated programmable interrupt controller (IPIC), including a definition of the external signals and their functions. Also, the configuration, control, and status registers are described in this chapter. Note that individual chapters in this reference manual describe specific initialization aspects for each individual block.

### 8.1 Introduction

This chapter describes the IPIC interrupt protocol, various types of interrupt sources controlled by the IPIC unit, and the IPIC registers with some programming guidelines. The interrupt controller provides interrupt management that is responsible for receiving hardware-generated interrupts from different sources (both internal and external). It also prioritizes and delivers the interrupts to the CPU for servicing. The IPIC prioritizes and manages interrupts from the following controller units:

- DDR memory controller (DDR2)
- Enhanced local bus memory controller (eLBC)
- DMA controller (DMA)
- Dual three-speed Ethernet controllers (eTSEC1 and eTSEC2) with IEEE 1588 support
- DUART communication module (DUART)
- USB 2.0 dual role controller (USB DR)
- System bus arbiter (SBA)
- Periodic interval timer (PIT)
- Real time clock timer (RTC ALR and RTC SEC)
- Global timer (GTM1) with 4 instances as GTM1\_1, GTM1\_2, GTM1\_3, and GTM1\_4
- Software watchdog timer (WDT)
- Two I<sup>2</sup>C controller (I<sup>2</sup>C1 and I<sup>2</sup>C2)
- SPI controller (SPI)
- General-purpose I/O controller (GPIO)
- External pins ( $\overline{\text{IRQ0}}$ ,  $\overline{\text{IRQ1}}$ ,  $\overline{\text{IRQ2}}$ , and  $\overline{\text{IRQ3}}$ )
- PCI Express controller
- Enhanced secure digital host controller (eSDHC)
- Message shared interrupt (MSI)

The interrupt sources controlled by the IPIC unit cause exceptions in the processor core. The internal interrupt (*int*) signal is the main interrupt output from the IPIC to the core and it causes the regular interrupt exception. The *cint* signal is the critical interrupt output from the IPIC to the processor core and causes the

critical interrupt exception. The  $\overline{smi}$  signal is the system management interrupt output from the IPIC to the processor core and causes the system management interrupt exception. The machine check exception is caused by the internal  $\overline{mcp}$  signal generated by the IPIC, informing the host processor of error conditions, assertion of the external  $\overline{IRQ0}$  machine-check request (enabled when  $SEMSR[SIRQ0] = 1$ ), and other conditions.

Table 8-1 shows the relationship of the various functional blocks and external signals of the device to the IPIC unit.

The IPIC receives interrupt request signals from the following two sources:

- External to the integrated device
- Internal to the integrated device

The unit selects the highest priority interrupt from all current interrupts and forwards it to the internal processor core, or off-chip for external servicing.

The IPIC also manages an internal non-maskable machine-check processor ( $\overline{mcp}$ ) signal and the interrupt generated by the off-chip interrupt sources ( $\overline{IRQ0}$ ,  $\overline{IRQ1}$ ,  $\overline{IRQ2}$ , and  $\overline{IRQ3}$ ).

The interrupt router of the IPIC monitors the outputs of the internal configuration registers. When the priority is highest in one of the received interrupt signals, the IPIC sets the corresponding bit in one of the following registers:

- System internal interrupt pending register (SIPNR)
- System external interrupt pending register (SEPNR)

If the interrupt is not masked, the IPIC asserts the  $\overline{int}$  signal to indicate an interrupt request to the processor. When the processor is running the specific  $\overline{int}$ ,  $\overline{cint}$ , or  $\overline{smi}$  interrupt handler code, the processor must vectorize the external interrupt handler by explicitly (in software) reading the corresponding interrupt vector register (SIVCR, SCVCR or SMVCR). In response to this read, the IPIC unit returns the vector (associated with the interrupt source) to the interrupt handler routine. In addition, the handler can vectorize different branches of interrupt handling.

Figure 8-1 shows the interrupt sources block diagram.

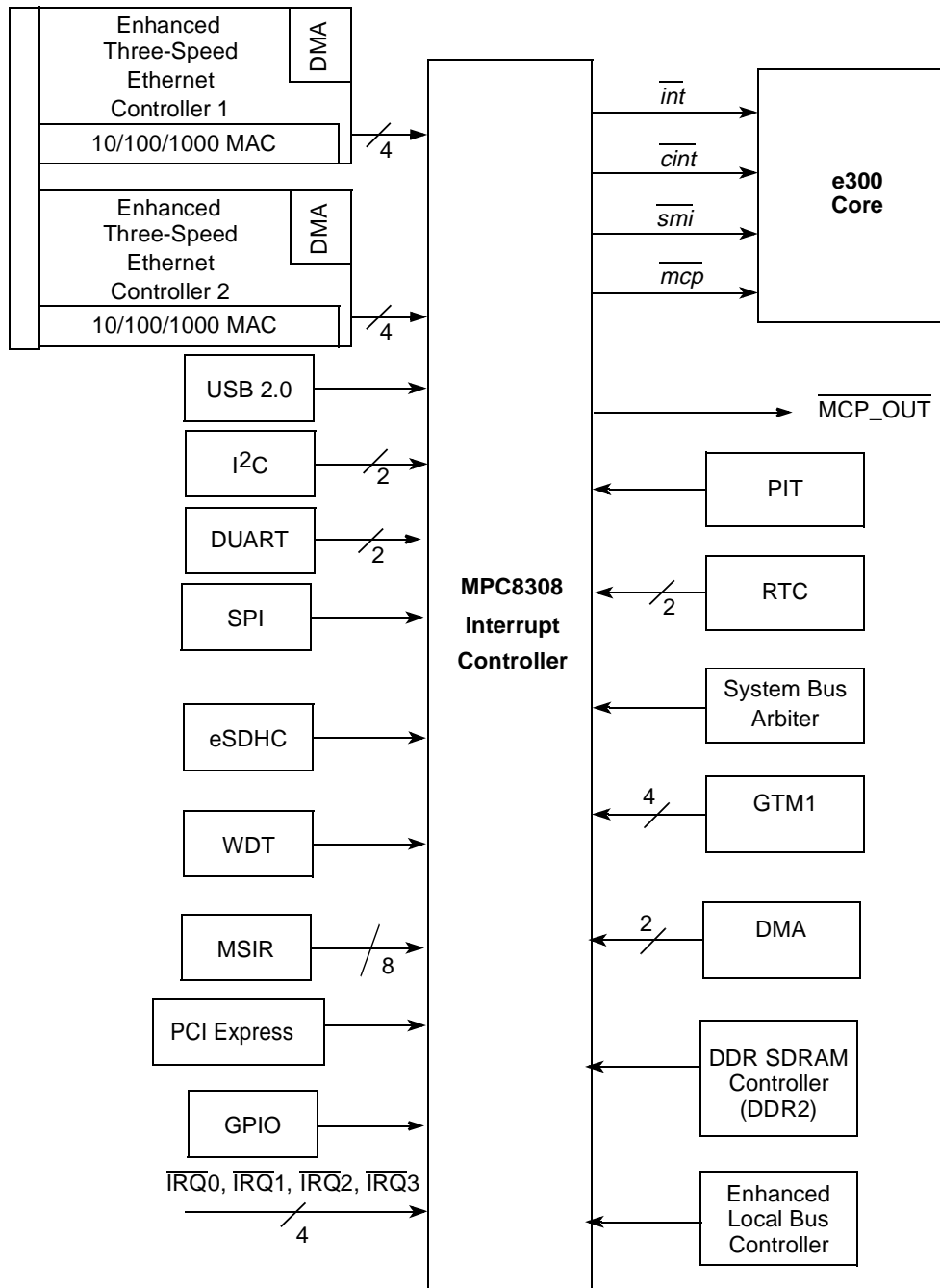


Figure 8-1. Interrupt Sources Block Diagram

The IPIC receives the following types of interrupts:

- External interrupt—triggered by the off-chip signals ( $\overline{IRQn}$ ) listed in Table 8-1
- Internal interrupts—on-chip interrupts, triggered by the sources listed in Table 8-7 and Table 8-9

- External and internal non-maskable machine check conditions, signaled by the sources listed in [Table 8-23](#) through  $\overline{mcp}$

The interrupt controller provides the ability to mask each interrupt source. Any source that can be caused by multiple events are also maskable.

When the IPIC receives an internal or external interrupt, its configuration register is checked to determine if it should be serviced as a normal external interrupt by the processor core (through the  $\overline{int}$  signal) or if the incoming interrupt has been configured as a critical or system management interrupt, the IPIC completes the processing of the interrupt by asserting  $\overline{cint}$  or  $\overline{smi}$  to the core. The assertion of the  $\overline{cint}$  or  $\overline{smi}$  signal to the core causes the interrupt to be serviced as a critical or a system management interrupt, respectively.

## 8.2 Features

The IPIC unit implements the following features:

- Support for external and internal discrete vectorized interrupt sources
- Support for external and internal non-maskable machine check conditions, signaled by  $\overline{mcp}$
- Programmable highest priority request (can be programmed to support a critical ( $\overline{cint}$ ) or system management interrupt ( $\overline{smi}$ ) type)
- Two programmable priority mixed groups of four on-chip and four external interrupt signals with two priority schemes for each group: grouped and spread
- Four programmable priority internal groups of eight on-chip interrupt signals with two priority schemes for each group: grouped and spread
- Two highest priority interrupts from each group can be programmed to support a critical or system management interrupt type
- External and internal interrupts directed to host processor
- Unique vector number for each interrupt source

## 8.3 Modes of Operation

The IPIC unit can operate in the core enable or core disable mode.

### 8.3.1 Core Enable Mode

In core enable mode, all internal interrupts are routed to and from the IPIC; the interrupts are sent to the e300 core. In this mode all machine check interrupts are gathered by the IPIC unit and sent to the e300 core.

### 8.3.2 Core Disable Mode

MPC8308 supports core disable mode only for debug purposes. The e300 core may be put in Core Disable mode by RCW. In this mode, JTAG will have access to the registers for reads and writes.

## 8.4 External Signal Description

The following sections provide an overview and detailed descriptions of the IPIC signals.

### 8.4.1 Overview

The device has 4 distinct external interrupt request input signals ( $\overline{\text{IRQ0}}$ ,  $\overline{\text{IRQ1}}$ ,  $\overline{\text{IRQ2}}$  and  $\overline{\text{IRQ3}}$ ). The IPIC interface signals are listed in [Table 8-1](#).

**Table 8-1. IPIC Signal Properties**

Name	Port	Function	I/O	Reset	Requires Pull Up
$\overline{\text{IRQ0}}$ , $\overline{\text{IRQ1}}$ , $\overline{\text{IRQ2}}$ , $\overline{\text{IRQ3}}$	$\overline{\text{IRQ0}}$ , $\overline{\text{IRQ1}}$ , $\overline{\text{IRQ2}}$ , $\overline{\text{IRQ3}}$	External interrupts	I	—	Yes
MCP_OUT	MCP_OUT	Interrupt request output	O	Z	Yes

### 8.4.2 Detailed Signal Descriptions

[Table 8-2](#) provides detailed descriptions of the external IPIC signals.

**Table 8-2. IPIC External Signals—Detailed Signal Descriptions**

Signal	I/O	Description
$\overline{\text{IRQ0}}$ , $\overline{\text{IRQ1}}$ , $\overline{\text{IRQ2}}$ , $\overline{\text{IRQ3}}$	I	Interrupt request 0, 1, 2, and 3. The sense (level or edge) of each of these signals is programmable. All of these inputs can be driven completely asynchronously.
		<b>State Meaning</b> Asserted—When an external interrupt request signal is asserted the priority is checked by the IPIC unit, and the interrupt is conditionally passed to the processor. Negated—There is no incoming interrupt from that source.
		<b>Timing</b> Assertion—All of these inputs can be asserted completely asynchronously. Negation—Interrupts programmed as level-sensitive must remain asserted until serviced.
MCP_OUT	OD	Non-maskable Interrupt (machine check) request out. Active-low, open drain. When the IPIC is programmed in core disable mode, this output reflects the <i>mcp</i> interrupts generated by on-chip sources. See <a href="#">Section 8.3</a> , “Modes of Operation.”
		<b>State Meaning</b> Asserted—At least one machine check interrupt is currently being signaled to the external system. Negated—Indicates no interrupt source currently routed to $\overline{\text{MCP\_OUT}}$ .
		<b>Timing</b> Because external interrupts are asynchronous with respect to the system clock, both assertion and negation of $\overline{\text{MCP\_OUT}}$ occurs asynchronously with respect to the interrupt source. All timing given here is approximate. Assertion—Internal interrupt source: 2 system bus clock cycles after interrupt occurs. External interrupt source: 4 cycles after interrupt occurs. Negation—Follows interrupt source negation with the following delay: Internal interrupt: 2 system bus clock cycles. External interrupt: 4 cycles.

## 8.5 Memory Map/Register Definition

The IPIC programmable register map occupies 256 bytes of memory-mapped space. Reading undefined portions of the memory map returns all zeros; writing has no effect.

All IPIC registers are 32 bits wide and they are located on 32-bit address boundaries. Software can perform byte, half-word, or word accesses to any IPIC registers. All addresses used in this chapter are offsets from the IPIC base, as defined in [Chapter 3, “Memory Map.”](#)

[Table 8-3](#) shows the memory map of the IPIC unit.

**Table 8-3. IPIC Register Address Map**

Offset	Register	Access	Reset Value	Section/ Page
<b>Integrated Programmable Interrupt Controller—Block Base Address 0x0_0700</b>				
0x000	System global interrupt configuration register (SICFR)	R/W	0x0000_0000	<a href="#">8.5.1/8-7</a>
0x004	System regular interrupt vector register (SIVCR)	R	0x0000_0000	<a href="#">8.5.2/8-8</a>
0x008	System internal interrupt pending register (SIPNR_H)	R	0x0000_0000	<a href="#">8.5.3/8-11</a>
0x00C	System internal interrupt pending register (SIPNR_L)	R	0x0000_0000	<a href="#">8.5.3/8-11</a>
0x010	System internal interrupt group A priority register (SIPRR_A)	R/W	0x0530_9770	<a href="#">8.5.4/8-13</a>
0x014	System internal interrupt group B priority register (SIPRR_B)	R/W	0x0530_9770	<a href="#">8.5.5/8-14</a>
0x018	System internal interrupt group C priority register (SIPRR_C)	R/W	0x0530_9770	<a href="#">8.5.6/8-15</a>
0x01C	System internal interrupt group D priority register (SIPRR_D)	R/W	0x0530_9770	<a href="#">8.5.7/8-16</a>
0x020	System internal interrupt mask register (SIMSR_H)	R/W	0x0000_0000	<a href="#">8.5.8/8-16</a>
0x024	System internal interrupt mask register (SIMSR_L)	R/W	0x0000_0000	<a href="#">8.5.8/8-16</a>
0x028	System internal interrupt control register (SICNR)	R/W	0x0000_0000	<a href="#">8.5.9/8-18</a>
0x02C	System external interrupt pending register (SEPNR)	R/W	Special	<a href="#">8.5.10/8-20</a>
0x030	System mixed interrupt group A priority register (SMPRR_A)	R/W	0x0530_9770	<a href="#">8.5.11/8-20</a>
0x034	System mixed interrupt group B priority register (SMPRR_B)	R/W	0x0530_9770	<a href="#">8.5.12/8-21</a>
0x038	System external interrupt mask register (SEMSR)	R/W	0x0000_0000	<a href="#">8.5.13/8-22</a>
0x03C	System external interrupt control register (SECNR)	R/W	0x0000_0000	<a href="#">8.5.14/8-23</a>
0x040	System error status register (SERSR)	R/W	0x0000_0000	<a href="#">8.5.15/8-24</a>
0x044	System error mask register (SERMR)	R/W		<a href="#">8.5.16/8-25</a>
0x048	System error control register (SERCR)	R/W	0x0000_0000	<a href="#">8.5.17/8-26</a>
0x04C	System external interrupt polarity control register (SEPCR)	R/W	0x0000_0000	<a href="#">8.5.18/8-26</a>
0x04F	Reserved	—	—	—
0x050	System internal interrupt force register (SIFCR_H)	R/W	0x0000_0000	<a href="#">8.5.19/8-27</a>
0x054	System internal interrupt force register (SIFCR_L)	R/W	0x0000_0000	<a href="#">8.5.19/8-27</a>
0x058	System external interrupt force register (SEFCR)	R/W	0x0000_0000	<a href="#">8.5.20/8-28</a>
0x05C	System error force register (SERFR)	R/W	0x0000_0000	<a href="#">8.5.21/8-29</a>
0x060	System critical interrupt vector register (SCVCR)	R	0x0000_0000	<a href="#">8.5.22/8-29</a>



Table 8-3. IPIC Register Address Map (continued)

Offset	Register	Access	Reset Value	Section/ Page
0x064	System management interrupt vector register (SMVCR)	R	0x0000_0000	8.5.23/8-30
0x068–0x0BF	Reserved	—	—	—

### 8.5.1 System Global Interrupt Configuration Register (SICFR)

SICFR, shown in Figure 8-2, defines the highest priority interrupt and whether interrupts are grouped or spread in the priority table.

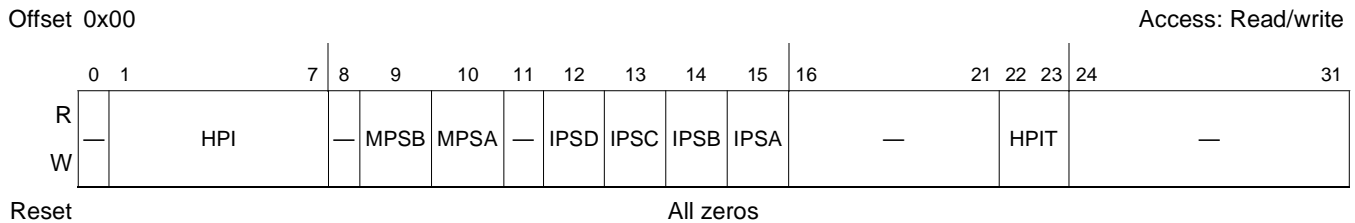


Figure 8-2. System Global Interrupt Configuration Register (SICFR)

Table 8-4 defines the bit fields of SICFR.

Table 8-4. SICFR Field Descriptions

Bits	Name	Description
0	—	Write ignored, read = 0
1–7	HPI	Highest priority interrupt. Specifies the 7-bit unique interrupt number/vector (see Table 8-6) of the single interrupt controller interrupt source that is advanced to the highest priority in the IPIC priority table (see Table 8-34). HPI can be modified dynamically.
8	—	Write ignored, read = 0
9	MPSB	Mixed interrupts priority scheme for group B. Selects the relative MIXB priority scheme. It cannot be changed dynamically. 0 Grouped. The MIXBs are grouped by priority at the top of the table. 1 Spread. The MIXBs are spread by priority in the table.
10	MPSA	Mixed interrupts priority scheme for group A. Selects the relative MIXA priority scheme. It cannot be changed dynamically. 0 Grouped. The MIXAs are grouped by priority at the top of the table. 1 Spread. The MIXAs are spread by priority in the table.
11	—	Write ignored, read = 0
12	IPSD	Internal interrupts priority scheme for group D. Selects the relative SYSD priority scheme. It cannot be changed dynamically. 0 Grouped. The SYSDs are grouped by priority at the top of the table. 1 Spread. The SYSDs are spread by priority in the table.
13	IPSC	Internal interrupts priority scheme for group C. Selects the relative SYSC priority scheme. It cannot be changed dynamically. 0 Grouped. The SYSCs are grouped by priority at the top of the table. 1 Spread. The SYSCs are spread by priority in the table.

Table 8-4. SICFR Field Descriptions (continued)

Bits	Name	Description
14	IPSB	Internal interrupts priority scheme for group B. Selects the relative SYSB priority scheme. It cannot be changed dynamically. 0 Grouped. The SYSBs are grouped by priority at the top of the table. 1 Spread. The SYSBs are spread by priority in the table.
15	IPSA	Internal interrupts priority scheme for group A. Selects the relative SYSA priority scheme. It cannot be changed dynamically. 0 Grouped. The SYSAs are grouped by priority at the top of the table. 1 Spread. The SYSAs are spread by priority in the table.
16–21	—	Write ignored, read = 0
22–23	HPIT	HPI priority position IPIC output interrupt type. Defines which type of IPIC output interrupt signal ( $\overline{int}$ , $\overline{cint}$ , or $\overline{smi}$ ) asserts its request to the core in the HPI priority position. These bits cannot be changed dynamically. (If software really wants to change it, it has to make sure the corresponding interrupt source is masked or it won't happen during the change). The definition of HPIT is as follows: 00 $\overline{int}$ request is asserted to the core for HPI. 01 $\overline{smi}$ request is asserted to the core for HPI. 10 $\overline{cint}$ request is asserted to the core for HPI. 11 Reserved.
24–31	—	Write ignored, read = 0

### 8.5.2 System Global Interrupt Vector Register (SIVCR)

SIVCR, shown in Figure 8-3, contains a 7-bit code (Table 8-5) representing the regular unmasked interrupt source (INT) of the highest priority level.

**NOTE**

Note that in core disabled mode the user should use SIVCR only in order to read an updated interrupt vector (SCVCR and SMVCR should not be used).

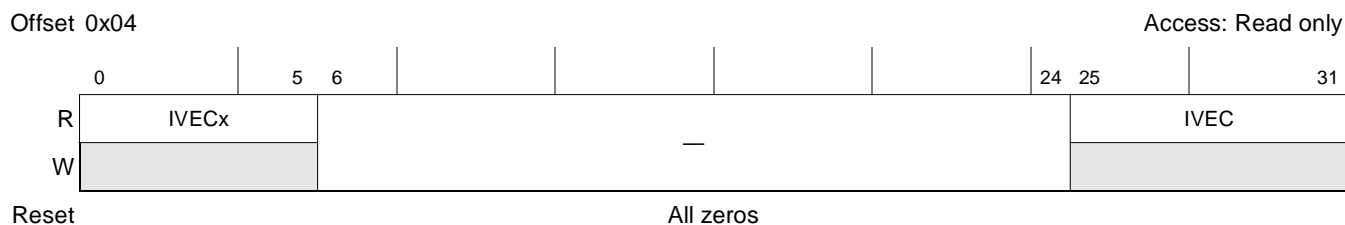


Figure 8-3. System Global Interrupt Vector Register (SIVCR)

Table 8-5 defines the bit fields of SIVCR.

**Table 8-5. SIVCR Field Descriptions**

Bits	Name	Description
0–5	IVECx	Backward (MPC8260) compatible regular interrupt vector. Specifies a 6-bit unique number of the IPIC's highest priority regular interrupt source, pending to the core. When a regular interrupt request occurs, SIVCR can be read. If there are multiple regular interrupt sources, SIVCR latches the highest priority regular interrupt. Note that IVECx field correctly reflects only the first 64 interrupt vectors (See Table 8-6 for details). The value of SIVCR cannot change while it is being read.
6–24	—	Write ignored, read = 0
25–31	IVEC	Regular interrupt vector. Specifies a 7-bit unique number of the IPIC's highest priority regular interrupt source, pending to the core. Note that the when a regular interrupt request occurs, SIVCR can be read. If there are multiple regular interrupt sources, SIVCR latches the highest priority regular interrupt. Note that the IVEC field correctly reflects all interrupt vectors (see Table 8-6 for details). The value of SIVCR cannot change while it is being read.

Table 8-6 shows the definition of IVEC.

**Table 8-6. IVEC/CVEC/MVEC Field Definition**

Interrupt ID Number	Interrupt Meaning	Interrupt Vector
0	Error (no interrupt)	0b000_0000
1	PCI Express CNT	0b000_0001
2	Reserved	0b000_0010
3	DMAC	0b000_0011
4	MSIR1	0b0000_0100
5–8	Reserved	0b000_0101–0b000_1000
9	UART1	0b000_1001
10	UART2	0b000_1010
11	Reserved	0b000_1011
12	eTSEC1 1588	0b000_1100
13	eTSEC2 1588	0b000_1101
14	I2C1	0b000_1110
15	I2C2	0b000_1111
16	SPI	0b001_0000
17	IRQ1	0b001_0001
18	IRQ2	0b001_0010
19	IRQ3	0b001_0011
20–31	Reserved	0b001_0100–0b001_1111
32	TSEC1 Tx	0b010_0000
33	TSEC1 Rx	0b010_0001
34	TSEC1 Err	0b010_0010

Table 8-6. IVEC/CVEC/MVEC Field Definition (continued)

Interrupt ID Number	Interrupt Meaning	Interrupt Vector
35	TSEC2 Tx	0b010_0011
36	TSEC2 Rx	0b010_0100
37	TSEC2 Err	0b010_0101
38	USB DR	0b010_0110
39–41	Reserved	0b010_0111–0b010_1001
42	eSDHC	0b010_1010
43–47	Reserved	0b010_1011–0x010_1111
48	IRQ0	0b011_0000
49–63	Reserved	0b011_0001–0b011_1111
64	RTC SEC	0b100_0000
65	PIT	0b100_0001
66	Reserved	0b100_0010
67	MSIR0	0b100_0011
68	RTC ALR	0b100_0100
69	Reserved	0b100_0101
70	SBA	0b100_0110
71	Reserved	0b100_0111
72	GTM1_4	0b100_1000
73	Reserved	0b100_1001
74	GPIO	0b100_1010
75	Reserved	0b100_1011
76	DDR	0b100_1100
77	LBC	0b100_1101
78	GTM1_2	0b100_1110
79–80	Reserved	0b100_1111–0b101_0000
81	MSIR2	0b101_0001
82	MSIR3	0b101_0010
83	Reserved	0b101_0011
84	GTM1_3	0b101_0100
85	Reserved	0b101_0101
86	MSIR4	0b101_0110
87	MSIR5	0b101_0111
88	MSIR6	0b101_1000
89	MSIR7	0b101_1001
90	GTM1_1	0b101_1010

Table 8-6. IVEC/CVEC/MVEC Field Definition (continued)

Interrupt ID Number	Interrupt Meaning	Interrupt Vector
91–93	Reserved	0b101_1011–0b101_1101
94	DMAC Err	0b101_1110
95–127	Reserved	0b101_1100–0b111_1111

### 8.5.3 System Internal Interrupt Pending Registers (SIPNR\_H and SIPNR\_L)

Each bit in SIPNR\_H and SIPNR\_L, shown in [Figure 8-4](#) and [Figure 8-5](#), is assigned an internal interrupt source (implemented bits are listed in [Table 8-7](#)). When an interrupt request is received, the interrupt controller sets the corresponding SIPNR bit. When a pending interrupt is handled, the user clears the SIPNR bit by clearing the corresponding event register bit.

Note that SIPNR bit positions are not changed according to relative priority.

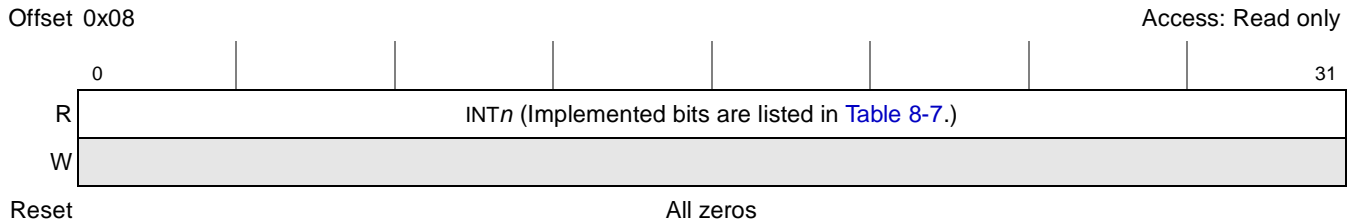


Figure 8-4. System Internal Interrupt Pending Register (SIPNR\_H)

[Table 8-7](#) lists implemented SIPNR\_H fields. Note that these field descriptions are also valid for SIFCR\_H and SIMSR\_H.

Table 8-7. SIPNR\_H/SIFCR\_H/SIMSR\_H Bit Assignments

Bits	Field
0	TSEC1 Tx
1	TSEC1 Rx
2	TSEC1 Err
3	TSEC2 Tx
4	TSEC2 Rx
5	TSEC2 Err
6	USB DR
7–9	—
10	eSDHC
11–15	—
16	PCI Express CNT
17	—
18	DMAC
19	MSIR1

**Table 8-7. SIPNR\_H/SIFCR\_H/SIMSR\_H Bit Assignments (continued)**

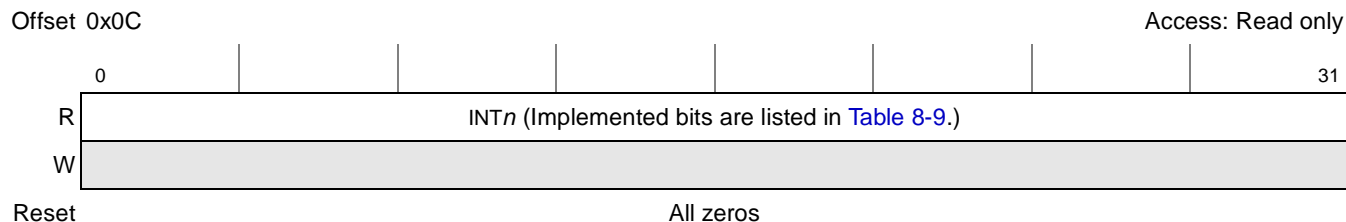
Bits	Field
20–23	—
24	UART1
25	UART2
26	—
27	eTSEC1 1588
28	eTSEC2 1588
29	I2C1
30	I2C2
31	SPI

Table 8-8 defines the bit fields of SIPNR\_H.

**Table 8-8. SIPNR\_H Field Descriptions**

Bits	Name	Description
0–31	INT <sub>n</sub>	Each implemented bit (listed in Table 8-7) corresponds to an internal interrupt source. When an interrupt is received, the interrupt controller sets the corresponding SIPNR bit. When a pending interrupt is handled, the user clears the SIPNR bit by clearing the corresponding event register bit. SIPNR bits are read only. Writing to this register has no effect. Note that the SIPNR bit positions are not changed according to their relative priority. For unimplemented bits, writes are ignored, read = 0.

SIPNR\_L is shown in Figure 8-5.



**Figure 8-5. System Internal Interrupt Pending Register (SIPNR\_L)**

Table 8-9 lists implemented SIPNR\_L fields. Note that these field assignments are also valid for SIFCR\_L and SIMSR\_L.

**Table 8-9. SIPNR\_L/SIFCR\_L/SIMSR\_L Bit Assignments**

Bits	Field
0	RTC SEC
1	PIT
2	—
3	MSIR0
4	RTC ALR
5	—
6	SBA

Table 8-9. SIPNR\_L/SIFCR\_L/SIMSR\_L Bit Assignments (continued)

Bits	Field
7	—
8	GTM1_4
9	—
10	GPIO
11	—
12	DDR
13	LBC
14	GTM1_2
15–16	—
17	MSIR2
18	MSIR3
19	—
20	GTM1_3
21	—
22	MSIR4
23	MSIR5
24	MSIR6
25	MSIR7
26	GTM1_1
27–29	—
30	DMAC Err
31	—

Table 8-10 defines the bit fields of SIPNR\_L.

Table 8-10. SIPNR\_L Field Descriptions

Bits	Name	Description
0–31	INT $n$	Each implemented bit (listed in Table 8-9) corresponds to an internal interrupt source. When an interrupt is received, the interrupt controller sets the corresponding SIPNR bit. When a pending interrupt is handled, the user clears the SIPNR bit by clearing the corresponding event register bit. SIPNR bits are read only. Writing to this register has no effect. Note that the SIPNR bit positions are not changed according to their relative priority. For unimplemented bits, writes are ignored, read = 0.

## 8.5.4 System Internal Interrupt Group A Priority Register (SIPRR\_A)

SIPRR\_A, shown in Figure 8-6, defines the priority between TSEC1 transmit request (TSEC1 Tx), TSEC1 receive request (TSEC1 Rx), TSEC1 transmit/receive error (TSEC1 Err), and internal interrupt signals.

For more information, see Section 8.6.3, “Internal Interrupts Group Relative Priority.”

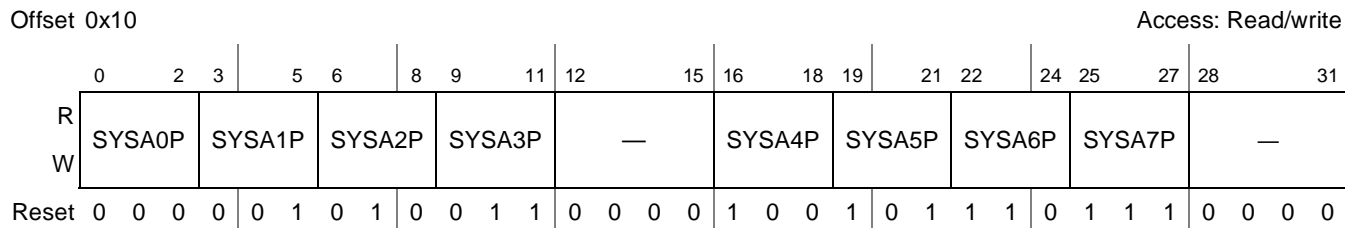


Figure 8-6. System Internal Interrupt Group A Priority Register (SIPRR\_A)

Table 8-11 defines the bit fields of SIPRR\_A.

Table 8-11. SIPRR\_A Field Descriptions

Bits	Name	Description
0–2	SYSA0P	SYSA0 priority order. Defines which interrupt source asserts its request in the SYSA0 priority position. The user should not program the same code to multiple priority positions (0–7). These bits can be changed dynamically. The definition of SYSA0P is as follows: 000 TSEC1 Tx asserts its request in the SYSA0 position. 001 TSEC1 Rx asserts its request in the SYSA0 position. 010 TSEC1 Err asserts its request in the SYSA0 position. 011 TSEC2 Tx asserts its request in the SYSA0 position. 100 TSEC2 Rx asserts its request in the SYSA0 position. 101 TSEC2 Err asserts its request in the SYSA0 position. 110 USB DR asserts its request in the SYSA0 position. 111 Reserved
3–11, 16–27	SYSA1P–SYSA7P	Same as SYSA0P, but for SYSA1P–SYSA7P.
12–15, 28–31	—	Write ignored, read = 0

### 8.5.5 System Internal Interrupt Group B Priority Register (SIPRR\_B)

The system internal interrupt group B priority register (SIPRR\_B), shown in Figure 8-7, defines the priority between internal interrupt signals.

For more information, see Section 8.6.3, “Internal Interrupts Group Relative Priority.”

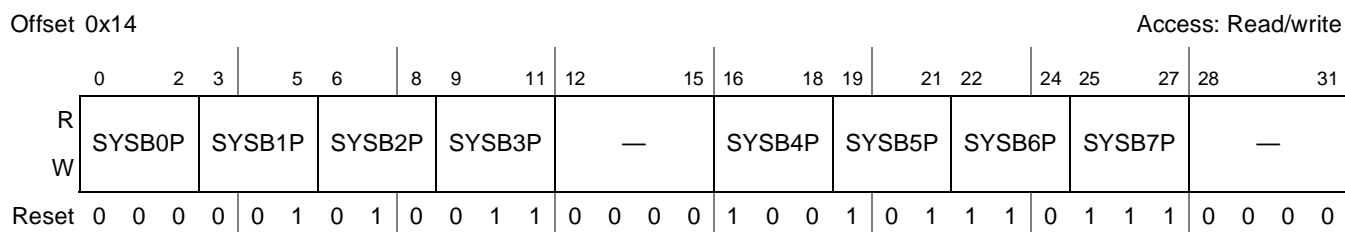


Figure 8-7. System Internal Interrupt Group B Priority Register (SIPRR\_B)



Table 8-12 defines the bit fields of SIPRR\_B.

**Table 8-12. SIPRR\_B Field Descriptions**

Bits	Name	Description
0–2	SYSB0P	SYSB0 Priority order. Defines which interrupt source asserts its request in the SYSB0 priority position. The user should not program the same code to more than one priority position (0–7). These bits can be changed dynamically. The definition of SYSB0P is shown as follows: 000–001 Reserved 010 eSDHC asserts its request in the SYSB0 position. 011–111 Reserved
3–11, 16–27	SYSB1P–SYSB7P	Same as SYSB0P, but for SYSB1P–SYSB7P.
12–15, 28–31	—	Write ignored, read = 0

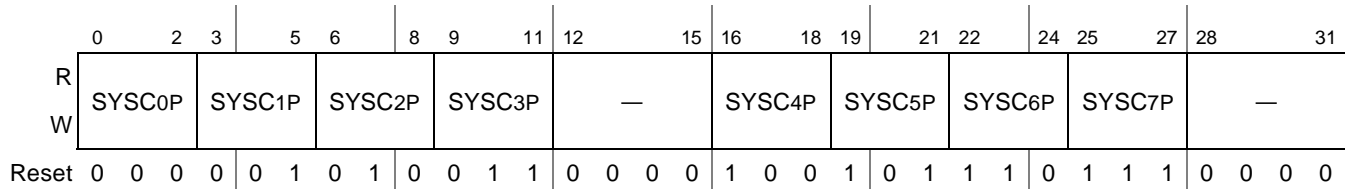
### 8.5.6 System Internal Interrupt Group C Priority Register (SIPRR\_C)

The system internal interrupt group C priority register (SIPRR\_C), shown in Figure 8-8, defines the priority between internal interrupt signals.

For more information about interrupt priorities, see Section 8.6.3, “Internal Interrupts Group Relative Priority.”

Offset 0x18

Access: Read/write



**Figure 8-8. System Internal Interrupt Group C Priority Register (SIPRR\_C)**

Table 8-13 defines the bit fields of SIPRR\_C.

**Table 8-13. SIPRR\_C Field Descriptions**

Bits	Name	Description
0–2	SYSC0P	SYSC0 priority order. Defines which interrupt source asserts its request in the SYSC0 priority position. The user should not program the same code to more than one priority position (0–7). These bits can be changed dynamically. The definition of SYSC0P is shown as follows: 000 PEX1 CNT asserts its request in the SYSC0 position. 001 Reserved 010 DMAC asserts its request in the SYSC0 position. 011 MSIR1 asserts its request in the SYSC0 position 100–111 Reserved
3–11, 16–27	SYSC1P–SYSC7P	Same as SYSC0P, but for SYSC1P–SYSC7P.
12–15, 28–31	—	Write ignored, read = 0

### 8.5.7 System Internal Interrupt Group D Priority Register (SIPRR\_D)

SIPRR\_D, shown in Figure 8-9, defines the priority among the interrupt sources listed in Table 8-14.

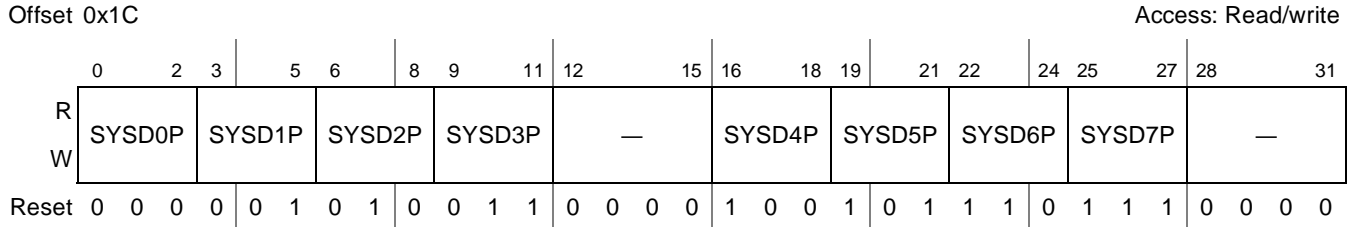


Figure 8-9. System Internal Interrupt Group D Priority Register (SIPRR\_D)

Table 8-14 defines the bit fields of SIPRR\_D.

Table 8-14. SIPRR\_D Field Descriptions

Bits	Name	Description
0–2	SYSD0P	SYSD0 priority order. Defines which interrupt source asserts its request in the SYSD0 priority position. The user should not program the same code to more than one priority position (0–7). These bits can be changed dynamically. SYSD0P is defined as follows: 000 UART1 asserts its request in the SYSD0 position. 001 UART2 asserts its request in the SYSD0 position. 010 Reserved 011 eTSEC1 1588 asserts its request in the SYSD0 position. 100 eTSEC2 1588 asserts its request in the SYSD0 position. 101 I2C1 asserts its request in the SYSD0 position. 110 I2C2 asserts its request in the SYSD0 position. 111 SPI asserts its request in the SYSD0 position.
3–11, 16–27	SYSD1P–SYSD7P	Same as SYSD0P, but for SYSD1P–SYSD7P.
12–15, 28–31	—	Write ignored, read = 0

### 8.5.8 System Internal Interrupt Mask Register (SIMSR\_H and SIMSR\_L)

Each implemented bit in SIMSR\_H and SIMSR\_L, shown in Figure 8-10 and Figure 8-11, corresponds to an internal interrupt source. The user masks an interrupt by clearing the corresponding SIMSR bit. When an interrupt request occurs, the corresponding SIPNR bit is set, regardless of the SIMSR bit. However, if the corresponding SIMSR bit is cleared, no interrupt request is passed to the core.

When simultaneously an SIMSR bit is cleared by the user and corresponding interrupt source requests an interrupt service, the request stops. If the user sets the SIMSR bit later, the core processes any pending corresponding interrupt requests according to its priority.



**Figure 8-10. System Internal Interrupt Mask Register (SIMSR\_H)**

[Table 8-15](#) defines the bit fields of SIMSR\_H.

**Table 8-15. SIMSR\_H Field Descriptions**

Bits	Name	Description
0–31	INT $n$	<p>Each implemented bit (listed in <a href="#">Table 8-7</a>) corresponds to an external interrupt source. The user masks an interrupt by clearing the corresponding SIMSR bit. An interrupt is unmasked (enable) by setting the corresponding SIMSR bit. The SIMSR can be read by the user at any time.</p> <p><b>Note:</b></p> <ul style="list-style-type: none"> <li>• SIMSR bit positions do not change according to their relative priority.</li> <li>• The user can clear pending register bits that were set by multiple interrupt events only by clearing all unmasked events in the corresponding event register.</li> <li>• If an SIMSR bit is masked at the same time that the corresponding SIPNR bit causes an interrupt request to the core, the error vector is issued (if no other interrupts are pending). Thus, the user should always include an error vector routine, even if it contains only an <code>rfi</code> instruction. The error vector cannot be masked.</li> </ul> <p>Unimplemented bits, shown as reserved in <a href="#">Table 8-7</a>, are ignored on writes; read = 0.</p>

[Figure 8-11](#) shows SIMSR\_L.



**Figure 8-11. System Internal Interrupt Mask Register (SIMSR\_L)**

Table 8-16 defines the bit fields of SIMSR\_L.

Table 8-16. SIMSR\_L Field Descriptions

Bits	Name	Description
0–31	INT <sub>n</sub>	<p>Each implemented bit (listed in Table 8-9) corresponds to an external interrupt source. The user masks an interrupt by clearing the corresponding SIMSR bit. An interrupt is unmasked (enabled) by setting the corresponding SIMSR bit. The SIMSR can be read by the user at any time.</p> <p><b>Note:</b></p> <ul style="list-style-type: none"> <li>SIMSR bit positions are not changed according to their relative priority.</li> <li>The user can clear pending register bits that were set by multiple interrupt events only by clearing all unmasked events in the corresponding event register.</li> <li>If an SIMSR bit is masked at the same time that the corresponding SIPNR bit causes an interrupt request to the core, the error vector is issued (if no other interrupts are pending). Thus, the user should always include an error</li> </ul> <p>Unimplemented bits, shown as reserved in Figure 8-11, are ignored on writes; read = 0.</p>

### 8.5.9 System Internal Interrupt Control Register (SICNR)

SICNR, shown in Figure 8-12, defines the IPIC output interrupt type ( $\overline{int}$ ,  $\overline{cint}$ , or  $\overline{smi}$ ) in the SYSA0–SYSA1, SYSB0–SYSB1, SYSC0–SYSC1, and SYSD0–SYSD1 priority positions. All other priority positions assert  $\overline{int}$  to the core.

Note that in core disabled mode the user should use the  $\overline{int}$  output interrupt type (should not use  $\overline{cint}$  or  $\overline{smi}$  output interrupt types) to read an updated SIVCR.

Offset 0x28

Access: Read write

	0	1	2	3	4	7	8	9	10	11	12	15	16	17	18	19	20	23	24	25	26	27	28	31
R	SYSD0T	SYSD1T	—	—	SYSC0T	SYSC1T	—	—	—	—	SYSB0T	SYSB1T	—	—	—	—	SYSA0T	SYSA1T	—	—	—	—	—	—
W																								

Reset

All zeros

Figure 8-12. System Internal Interrupt Control Register (SICNR)

Table 8-17 defines the bit fields of SICNR.

Table 8-17. SICNR Field Descriptions

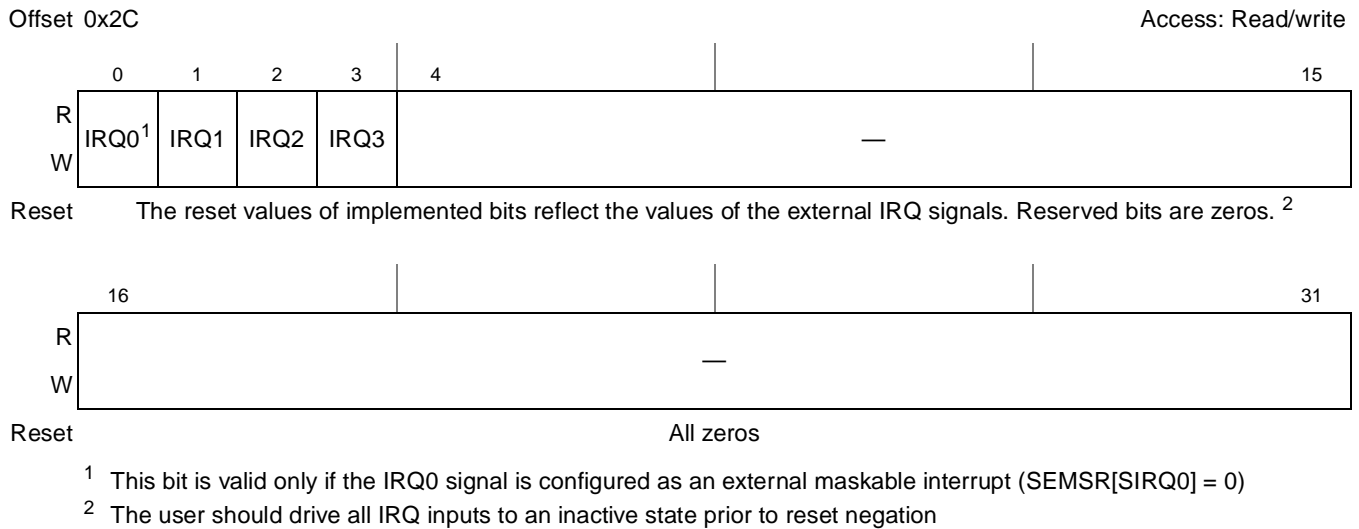
Bits	Name	Description
0–1	SYSD0T	<p>SYSD0 priority position IPIC output interrupt type. Defines which type of the IPIC output interrupt signal (<math>\overline{int}</math>, <math>\overline{cint}</math>, or <math>\overline{smi}</math>) asserts its request to the core in the SYSD0 priority position. These bits cannot be changed dynamically. (to change it, software must make sure the corresponding interrupt source is masked or it cannot happen during the change).</p> <p>The definition of SYSD0T is as follows:</p> <p>00 <math>\overline{int}</math> request is asserted to the core for SYSD0.                      01 <math>\overline{smi}</math> request is asserted to the core for SYSD0.                      10 <math>\overline{cint}</math> request is asserted to the core for SYSD0.                      11 Reserved</p>
2–3	SYSD1T	Same as SYSD0T, but for SYSD1T.
4–7	—	Write ignored, read = 0

Table 8-17. SICNR Field Descriptions (continued)

Bits	Name	Description
8–9	SYSC0T	<p>SYSC0 priority position IPIC output interrupt Type. Defines which type of the IPIC output interrupt signal (<math>\overline{int}</math>, <math>\overline{cint}</math>, or <math>\overline{smi}</math>) asserts its request to the core in the SYSC0 priority position. These bits can not be changed dynamically. (If s/w really wants to change it, it has to make sure the corresponding interrupt source is masked or it won't happen during the change).</p> <p>The definition of SYSC0T is as follows:</p> <p>00 <math>\overline{int}</math> request is asserted to the core for SYSC0.  01 <math>\overline{smi}</math> request is asserted to the core for SYSC0.  10 <math>\overline{cint}</math> request is asserted to the core for SYSC0.  11 Reserved</p>
10–11	SYSC1T	Same as SYSC0T, but for SYSC1T.
12–15	—	Write ignored, read = 0
16–17	SYSB0T	<p>SYSB0 priority position IPIC output interrupt Type. Defines which type of the IPIC output interrupt signal (<math>\overline{int}</math>, <math>\overline{cint}</math>, or <math>\overline{smi}</math>) asserts its request to the core in the SYSB0 priority position. These bits can not be changed dynamically. (If software really wants to change it, it has to make sure the corresponding interrupt source is masked or it does not happen during the change).</p> <p>The definition of SYSB0T is as follows:</p> <p>00 <math>\overline{int}</math> request is asserted to the core for SYSB0.  01 <math>\overline{smi}</math> request is asserted to the core for SYSB0.  10 <math>\overline{cint}</math> request is asserted to the core for SYSB0.  11 Reserved</p>
18–19	SYSB1T	Same as SYSB0T, but for SYSB1T.
20–23	—	Write ignored, read = 0
24–25	SYSA0T	<p>SYSA0 priority position IPIC output interrupt type. Defines which type of the IPIC output interrupt signal (<math>\overline{int}</math>, <math>\overline{smi}</math>, or <math>\overline{cint}</math>) asserts its request to the core in the SYSA0 priority position. These bits can not be changed dynamically. (If software really wants to change it, it has to make sure the corresponding interrupt source is masked or it does not happen during the change).</p> <p>The definition of SYSA0T is as follows:</p> <p>00 <math>\overline{int}</math> request is asserted to the core for SYSA0.  01 <math>\overline{smi}</math> request is asserted to the core for SYSA0.  10 <math>\overline{cint}</math> request is asserted to the core for SYSA0.  11 Reserved.</p>
26–27	SYSA1T	Same as SYSA0T, but for SYSA1T
28–31	—	Write ignored, read = 0

### 8.5.10 System External Interrupt Pending Register (SEPNR)

Each bit in the SEPNR, shown in Figure 8-13, corresponds to an external interrupt source. When an interrupt is received, the interrupt controller sets the corresponding SEPNR bit.



**Figure 8-13. System External Interrupt Pending Register (SEPNR)**

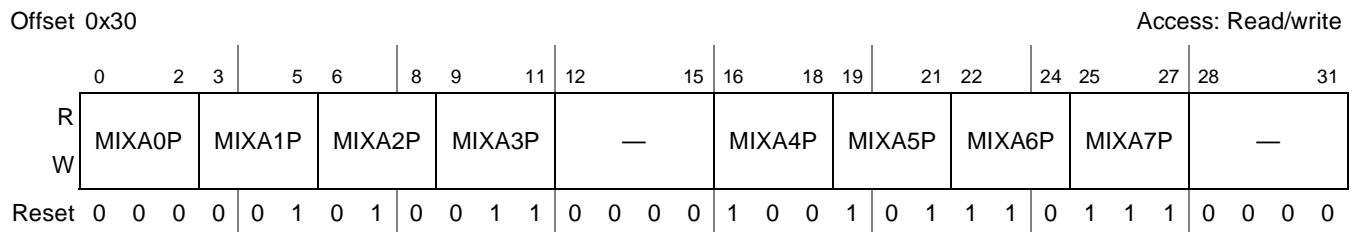
Table 8-18 defines the bit fields of SEPNR.

**Table 8-18. SEPNR Field Descriptions**

Bits	Name	Description
0, 1, 2, 3	IRQ <sub>n</sub>	Each bit corresponds to an external interrupt source. When an external interrupt is received, the interrupt controller sets the corresponding SEPNR bit. When a pending interrupt is handled, the user must clear the corresponding SEPNR bit. For level triggered cases, the software needs to cause the IRQ <sub>n</sub> to negate which automatically clears the bit in SEPNR. For edge-triggered cases, the software needs to clear the corresponding bit in SEPNR. SEPNR bits are cleared by writing ones to them. Because the user can only clear bits in this register, writing zeros to this register has no effect. Note that the SEPNR bit positions are not changed according to their relative priority.
4-31	—	Write ignored, read = 0

### 8.5.11 System Mixed Interrupt Group A Priority Register (SMPRR\_A)

The SMPRR\_A, shown in Figure 8-14, defines the priority among the sources listed in Table 8-19.



**Figure 8-14. System Mixed Interrupt Group A Priority Register (SMPRR\_A)**

Table 8-19 defines the bit fields of SMPRR\_A.

**Table 8-19. SMPRR\_A Field Descriptions**

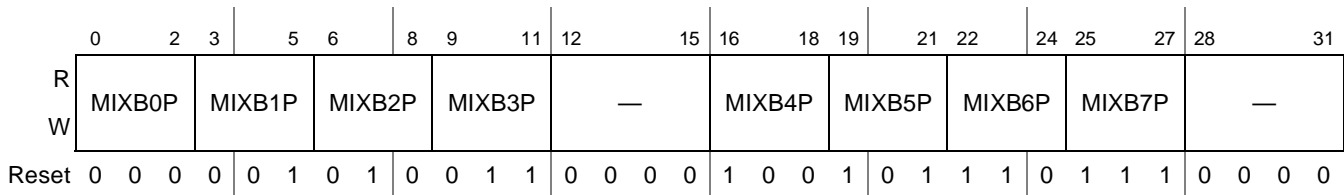
Bits	Name	Description
0–2	MIXA0P	MIXA0 priority order. Defines which interrupt source asserts its request in the MIXA0 priority position. The user must not program the same code to more than one priority position (0–7). These bits can be changed dynamically. The definition of MIXA0P is as follows: 000 RTC SEC asserts its request to the MIXA0 position. 001 PIT asserts its request to the MIXA0 position. 010 Reserved 011 MSIR0 asserts its request to the MIXA0 position. 100 IRQ0 asserts its request to the MIXA0 position. This field for MIXA0 position is valid (must not be ignored) if IRQ0 signal configured as an external maskable interrupt (SEMSR[SIRQ0] = 0). 101 IRQ1 asserts its request to the MIXA0 position. 110 IRQ2 asserts its request to the MIXA0 position. 111 IRQ3 asserts its request to the MIXA0 position.
3–11, 16–27	MIXA1P–MIXA7P	Same as MIXA0P, but for MIXA1P–MIXA7P.
12–15, 28–31	—	Write ignored, read = 0

### 8.5.12 System Mixed Interrupt Group B Priority Register (SMPRR\_B)

SMPRR\_B, shown in Figure 8-15, defines the priority among the sources listed in Table 8-20.

Offset 0x34

Access: Read/write



**Figure 8-15. System Mixed Interrupt Group B Priority Register (SMPRR\_B)**

Table 8-20 defines the bit fields of SMPRR\_B.

**Table 8-20. SMPRR\_B Field Descriptions**

Bits	Name	Description
0–2, 3–11, 16–27	MIXBnP	MIXBn priority order. Defines which interrupt source asserts its request in the MIXBn priority position. The user must not program the same code to more than one priority position (0–7). These bits can be changed dynamically. The definition of MIXBnP is as follows: 000 RTC ALR asserts its request to the MIXBn position. 001 Reserved 010 SBA asserts its request to the MIXBn position. 011–111 Reserved
12–15, 28–31	—	Write ignored, read = 0

### 8.5.13 System External Interrupt Mask Register (SEMSR)

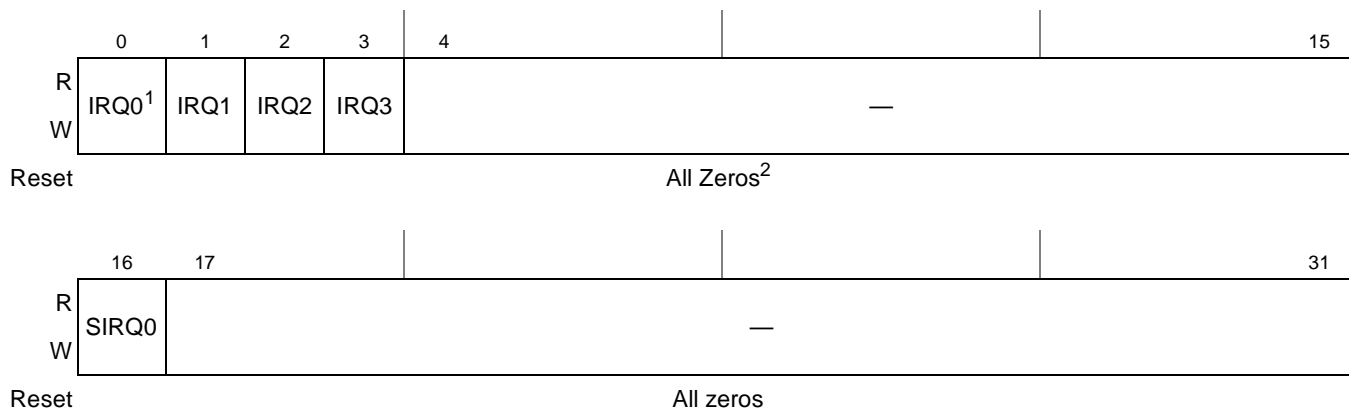
Each bit in the system external interrupt mask register (SEMSR), shown in Figure 8-13, corresponds to an external interrupt source. The user masks an interrupt by clearing the corresponding SEMSR bit. An interrupt is unmasked (enabled) by setting the corresponding SEMSR bit.

When an external interrupt request occurs, the corresponding SEP NR bit is set regardless of the setting of the corresponding SEMSR bit. However, if the corresponding SEMSR bit is cleared, no interrupt request is passed to the core.

When simultaneously an SEMSR bit is cleared by the user and an interrupt source requests an interrupt service, the request stops. If the user sets the SEMSR bit later, a previously pending interrupt request is processed by the core according to its assigned priority. SEMSR can be read by the user at any time.

Offset 0x38

Access: Read/write



<sup>1</sup> This bit is valid only if the IRQ0 signal is configured as an external maskable interrupt (SEMSR[SIRQ0] = 0)

<sup>2</sup> The user should drive all IRQ inputs to an inactive state prior to reset negation

**Figure 8-16. System External Interrupt Mask Register (SEMSR)**

Table 8-21 defines the bit fields of SEMSR.

**Table 8-21. SEMSR Field Descriptions**

Bits	Name	Description
0, 1, 2, 3	—	Each bit corresponds to an external interrupt source. The user masks an interrupt by clearing the SEMSR bit. An interrupt can be enabled by setting the corresponding SEMSR bit. SEMSR can be read by the user at any time. <b>Note:</b> <ul style="list-style-type: none"> <li>• SEMSR bit positions are not affected by their relative priority.</li> <li>• The user can clear pending register bits that were set by multiple interrupt events only by clearing all unmasked events in the corresponding event register.</li> <li>• If an SEMSR bit is masked at the same time that the corresponding SEP NR bit causes an interrupt request to the core, the error vector is issued (if no other interrupts pending). Thus, the user must always include an error vector routine, even if it contains only an rfi instruction. The error vector cannot be masked.</li> </ul>
4-15	—	Write ignored, read = 0



Table 8-21. SEMSR Field Descriptions (continued)

Bits	Name	Description
16	SIRQ0	Steer IRQ0. 0 IRQ0 is used as external interrupt request 1 IRQ0 is used as external MCP request
17–31	—	Write ignored, read = 0

### 8.5.14 System External Interrupt Control Register (SECNR)

SECNR, shown in Figure 8-17, defines the edge detect mode for external  $\overline{IRQ}_n$  interrupt signals and determines whether the corresponding  $\overline{IRQ}_n$  signal asserts an interrupt request upon either a high-to-low change or assertion on the pin. It also defines the IPIC output interrupt type ( $\overline{int}$ ,  $\overline{cint}$ , or  $\overline{smi}$ ) in the MIXA0–MIXA1 and MIXB0–MIXB1 priority positions.

Note that in core disabled mode of operation the user should use the  $\overline{int}$  output interrupt type (should not use  $\overline{cint}$  or  $\overline{smi}$  output interrupt types) in order to read an updated SIVCR.

Offset 0x3C

Access: Read/write

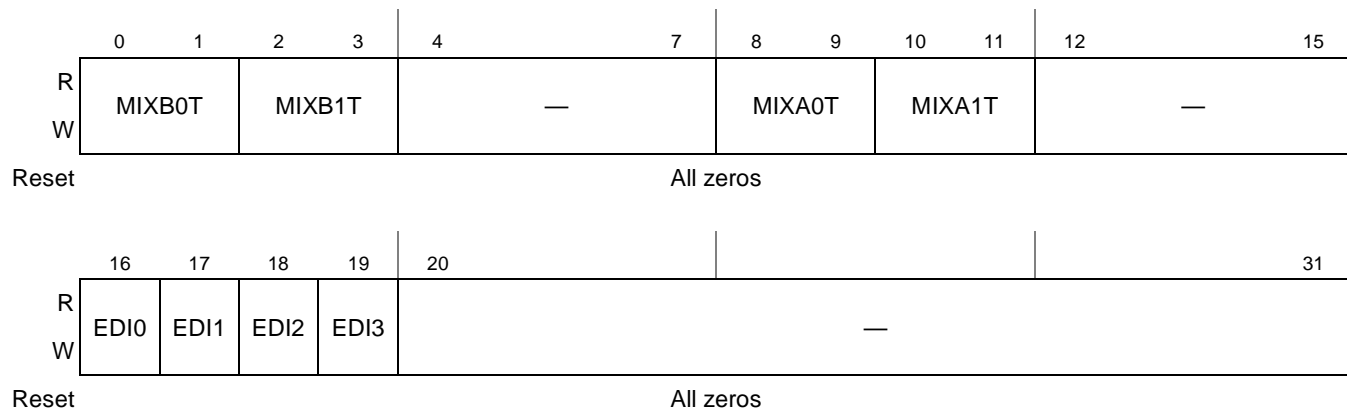


Figure 8-17. System External Interrupt Control Register (SECNR)

Table 8-22 defines the bit fields of SECNR.

Table 8-22. SECNR Field Descriptions

Bits	Name	Description
0–1	MIXB0T	MIXB0 priority position IPIC output interrupt type. Defines which type of the IPIC output interrupt signal ( $\overline{int}$ , $\overline{cint}$ , or $\overline{smi}$ ) asserts its request to the core in the MIXB0 priority position. These bits can be changed dynamically. The definition of MIXB0T is as follows: 00 $\overline{int}$ request is asserted to the core for MIXB0. 01 $\overline{smi}$ request is asserted to the core for MIXB0. 10 $\overline{cint}$ request is asserted to the core for MIXB0. 11 Reserved
2–3	MIXB1T	Same as MIXB0T, but for MIXB1T.
4–7	—	Write ignored, read = 0

**Table 8-22. SECNR Field Descriptions (continued)**

Bits	Name	Description
8–9	MIXA0T	MIXA0 priority position IPIC output interrupt Type. Defines which type of the IPIC output interrupt signal ( $\overline{int}$ , $\overline{cint}$ , or $\overline{smi}$ ) asserts its request to the core in the MIXA0 priority position. These bits can be changed dynamically. The definition of MIXA0T is as follows: 00 $\overline{int}$ request is asserted to the core for MIXA0. 01 $\overline{smi}$ request is asserted to the core for MIXA0. 10 $\overline{cint}$ request is asserted to the core for MIXA0. 11 Reserved
10–11	MIXA1T	Same as MIXA0T, but for MIXA1T.
12–15	—	Write ignored, read = 0
16–19	EDix	Each bit defines the edge detect mode for the external $\overline{IRQn}$ interrupt signals, determines whether the corresponding $\overline{IRQn}$ signal asserts an interrupt request upon either a high-to-low (high assertion for active high polarity) change or low assertion (high assertion for active high polarity) on the pin. The corresponding $\overline{IRQn}$ signal asserts an interrupt request as follows: 0 Low assertion (high assertion for active high polarity) on $\overline{IRQn}$ generates an interrupt request (level sensitive). 1 High-to-low (low-to-high for active high polarity) change on $\overline{IRQn}$ generates an interrupt request (edge sensitive).
20–31	—	Write ignored, read = 0

### 8.5.15 System Error Status Register (SERSR)

The bits in the SERSR, shown in [Figure 8-18](#), correspond to the external and internal non-maskable error source machine check (mcp) conditions listed in [Table 8-23](#). When an error interrupt signal is received, the interrupt controller sets the corresponding SERSR bit.



**Figure 8-18. System Error Status Register (SERSR)**

[Table 8-23](#) lists the implemented SERSR bits. Note that these field assignments are valid for SERMR and SERFR.

**Table 8-23. SERSR/SERMR/SERFR Bit Assignments**

Bits	Field
0	IRQ0 <sup>1</sup>
1	WDT
2	SBA
3–31	—

<sup>1</sup> This bit is valid only if the IRQ0 signal is configured as an external MCP interrupt (SEMSR[SIRQ0] = 1)

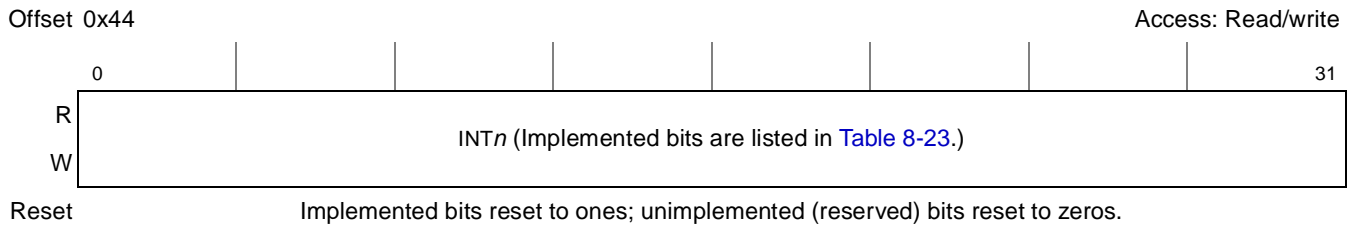
Table 8-24 defines the bit fields of SERSR.

**Table 8-24. SERSR Field Descriptions**

Bits	Name	Description
0–31	INT <sub>n</sub>	Each implemented bit in the SERSR, listed in Table 8-23, corresponds to an external and an internal error source (mcp). When an error interrupt signal is received, the interrupt controller sets the corresponding SERSR bit. SERSR bits are cleared by writing ones to them. Unmasked event register bits should be cleared before clearing SERSR bits. Because the user can only clear bits in this register, writing zeros to this register has no effect. SERSR bits are cleared by power-on reset. Subsequent soft and hard resets do not affect SERSR bit states. For unimplemented bits (listed as reserved in Table 8-23), writes are ignored, read = 0

### 8.5.16 System Error Mask Register (SERMR)

Each implemented bit in SERMR, shown in Figure 8-19, corresponds to an external and an internal  $\overline{mcp}$  source (MCP). The user masks an MCP by clearing and enables an interrupt by setting the corresponding SERMR bit. When a masked MCP occurs, the corresponding SERSR bit is set, regardless of the setting of the corresponding SERMR bit although no MCP request is passed to the core in this case. The SERMR can be read by the user at any time.



**Figure 8-19. System Error Mask Register (SERMR)**

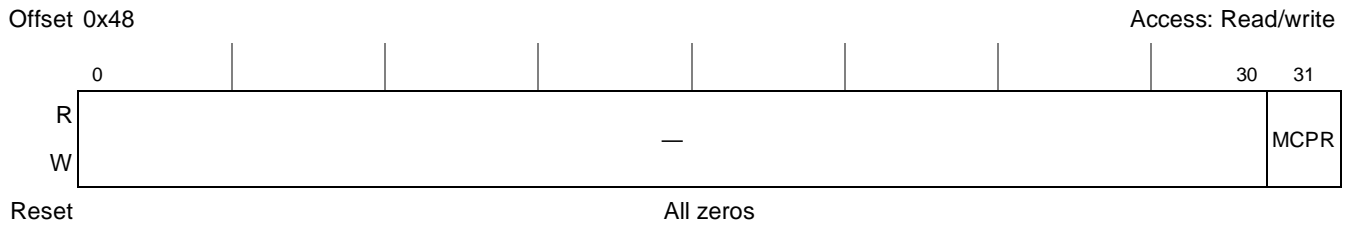
Table 8-25 defines the bit fields of SERMR.

**Table 8-25. SERMR Field Descriptions**

Bits	Name	Description
0–31	INT <sub>n</sub>	Each implemented SERMR bit, listed in Table 8-23, corresponds to an external and an internal MCP source. The user masks an MCP by clearing and enables an interrupt by setting the corresponding SERMR bit. When a masked MCP occurs, the corresponding SERSR bit is set, regardless of the setting of the SERMR bit although no MCP request is passed to the core. The SERMR can be read by the user at any time. Writes to unimplemented (reserved) bits are ignored; read = 0

## 8.5.17 System Error Control Register (SERCR)

SERCR, shown in Figure 8-20, defines the control bits that route MCP requests in core disable mode to  $\overline{\text{MCP\_OUT}}$



**Figure 8-20. System Error Control Register (SERCR)**

Table 8-26 defines the bit fields of SERCR.

**Table 8-26. SERCR Field Descriptions**

Bits	Name	Description
0–30	—	Write ignored, read = 0
31	MCPR	MCP route. Route MCP request to $\overline{\text{MCP\_OUT}}$ . 0 MCP is not available at $\overline{\text{MCP\_OUT}}$ 1 MCP routed to $\overline{\text{MCP\_OUT}}$ .

## 8.5.18 System External interrupt Polarity Control Register (SEPCR)

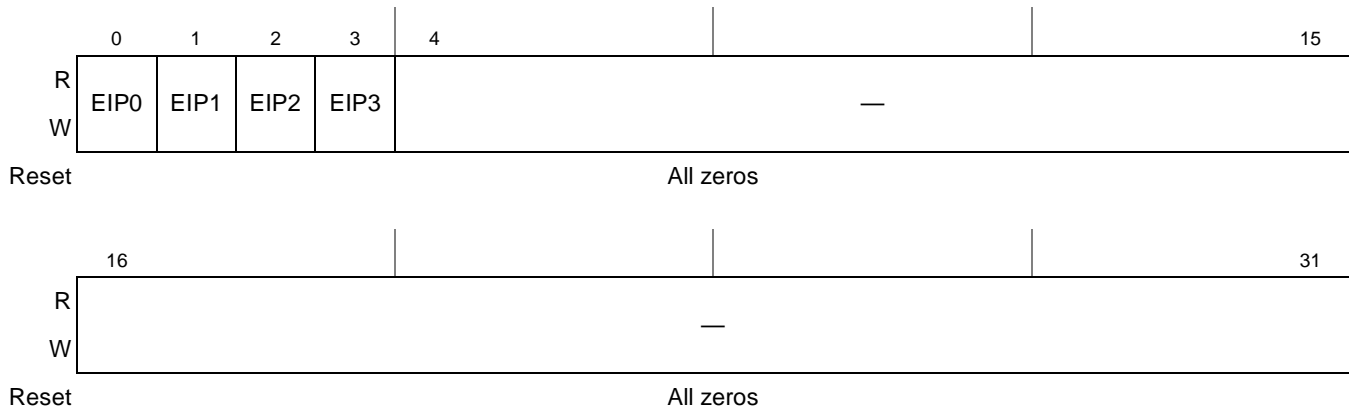
SEPCR, shown in Figure 8-21, defines the polarity for each one of the external  $\overline{\text{IRQ}}_n$  interrupt signals and determines whether the corresponding  $\overline{\text{IRQ}}_n$  signal is treated as active low or active high signal. The active low signals will assert an interrupt request upon either a high-to-low change or assertion (low state) on the pin. The active high signals assert an interrupt request upon either a low-to-high change or assertion (high state) on the pin. See Section 8.5.14, “System External Interrupt Control Register (SECNR),” on page 8-23 for more details.

### NOTE

Note that the  $\overline{\text{IRQ}}_n$  signals are overbarred although the SEPCR could be programmed to accept active high signals. The overbar should be ignored in this case.

Offset 0x4C

Access: Read/write



**Figure 8-21. System External Interrupt Polarity Control Register (SEPCR)**

Table 8-27 defines the bit fields of SEPCR.

**Table 8-27. SEPCR Field Descriptions**

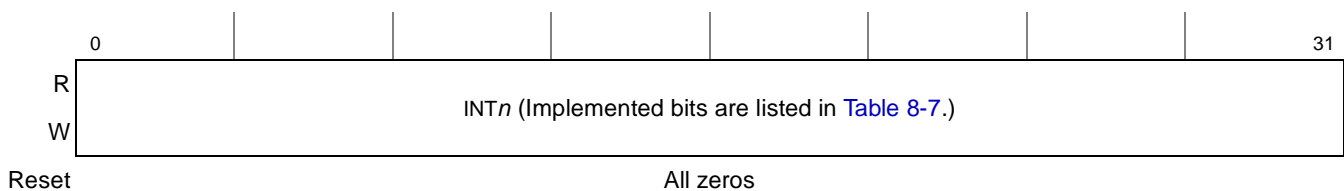
Bits	Name	Description
0–3	EIPx	Each bit defines the active state for the $\overline{IRQ}n$ interrupt signals. 0 Active Low. 1 Active High.
4–31	—	Write ignored, read = 0

### 8.5.19 System Internal Interrupt Force Registers (SIFCR\_H and SIFCR\_L)

Each implemented bit SIFCR\_H and SIFCR\_L, shown in Figure 8-22 and Figure 8-23, corresponds to an internal interrupt source. When a bit is set, the interrupt controller generates the corresponding interrupt (sets the corresponding SIPNR bit). The SIFCR can be read by the user at any time.

Offset 0x50

Access: Read/write



**Figure 8-22. System Internal Interrupt Force Register (SIFCR\_H)**

Table 8-28 defines the bit fields of SIFCR\_H.

**Table 8-28. SIFCR\_H Field Descriptions**

Bits	Name	Description
0–31	INTn	Each implemented bit, listed in Table 8-7, corresponds to an internal interrupt source. The user forces an interrupt by setting the corresponding SIFCRx bit. SIFCRn bit positions are not changed according to their relative priority. Writes to unimplemented (reserved) bits are ignored; read = 0

SIFCR\_L is shown in [Figure 8-23](#).



**Figure 8-23. System Internal Interrupt Force Register (SIFCR\_L)**

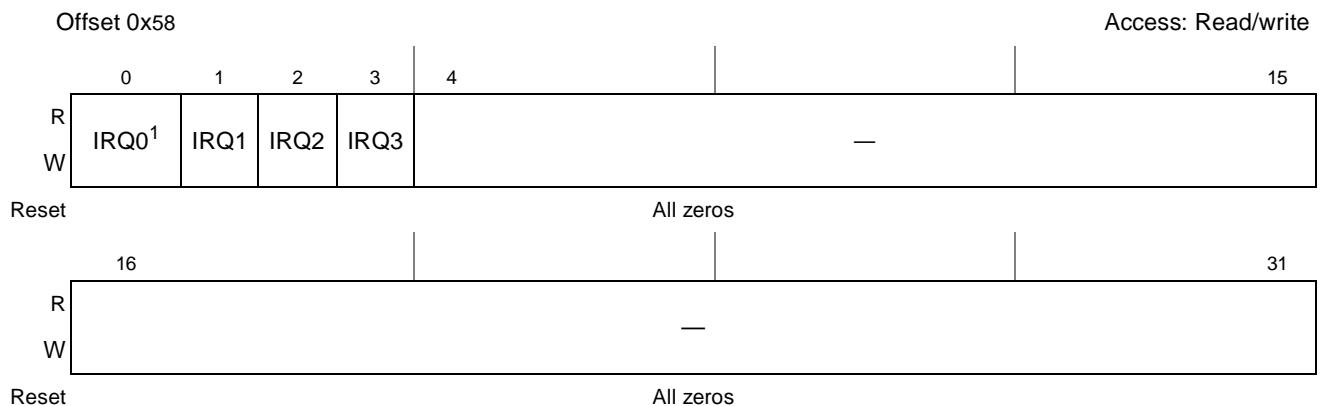
[Table 8-29](#) defines the bit fields of SIFCR\_L.

**Table 8-29. SIFCR\_L Field Descriptions**

Bits	Name	Description
0–31	INT $n$	Each implemented bit, listed in <a href="#">Table 8-9</a> , corresponds to an internal interrupt source. The user forces an interrupt by setting the corresponding SIFCR $x$ bit. SIFCR $x$ bit positions are not changed according to their relative priority. Writes to unimplemented (reserved) bits are ignored; read = 0

### 8.5.20 System External Interrupt Force Register (SEFCR)

Each implemented bit in SEFCR, shown in [Figure 8-24](#), corresponds to an external interrupt source. When a bit is set, the interrupt controller generates the corresponding external interrupt (sets the corresponding SEP $NR$  bit). SEFCR can be read by the user at any time.



<sup>1</sup> This bit is valid only if IRQ0 is configured as an external maskable interrupt (SEMSR[SIRQ0] = 0)

**Figure 8-24. System External Interrupt Force Register (SEFCR)**

Table 8-30 defines the bit fields of SEFCR.

**Table 8-30. SEFCR Field Descriptions**

Bits	Name	Description
0, 1, 2, 3	IRQ $n$	Each bit corresponds to an external interrupt source. The user forces an interrupt by setting the SEFCR bit. <b>Note:</b> SEFCR bit positions are not affected by their relative priority.
4–31	—	Write ignored, read = 0

### 8.5.21 System Error Force Register (SERFR)

Each bit in the system error force register (SERFR), shown in Figure 8-25, corresponds to an external MCP source. When a bit is set, the interrupt controller generates the corresponding MCP interrupt (sets the corresponding SERSR bit). The SERFR can be read by the user at any time.



**Figure 8-25. System Error Status Register (SERFR)**

Table 8-31 defines the bit fields of SERFR.

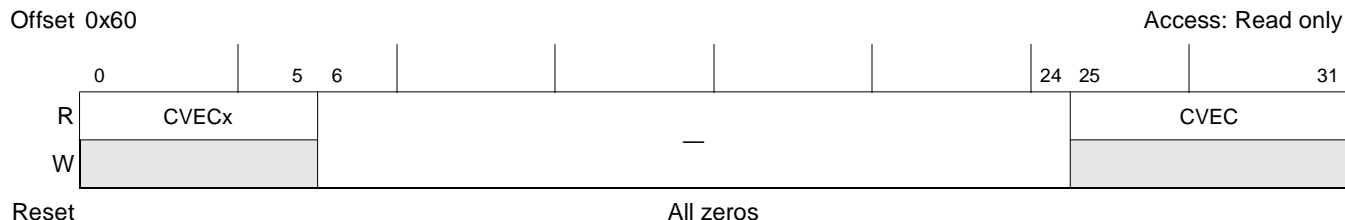
**Table 8-31. SERFR Field Descriptions**

Bits	Name	Description
0–31	INT $n$	Each implemented bit, listed in Table 8-23, corresponds to an external MCP source. The user forces an MCP by setting the SERFR bit. SERFR bit positions are not affected by their relative priority. Attempts to write to unimplemented (reserved) bits are ignored; read = 0

### 8.5.22 System Critical Interrupt Vector Register (SCVCR)

SCVCR, shown in Figure 8-26, contains a 7-bit code (Table 8-32) representing the unmasked critical interrupt ( $\overline{cint}$ ) source of the highest priority level.

Note that in core-disabled mode the user should use SIVCR only to read an updated interrupt vector (SCVCR should not be used).



**Figure 8-26. System Critical Interrupt Vector Register (SCVCR)**

Table 8-32 defines SCVCR bit fields.

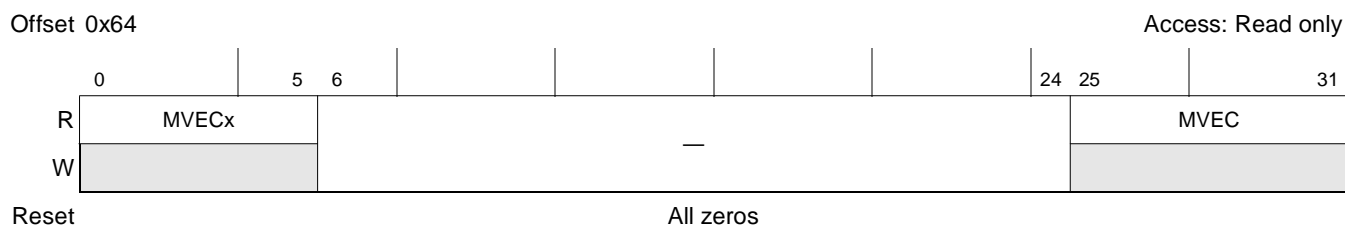
**Table 8-32. SCVCR Field Descriptions**

Bits	Name	Description
0–5	CVECx	Backward (MPC8260) compatible critical interrupt vector. Specifies a 6-bit unique number of the IPIC’s highest priority critical interrupt source, pending to the core. When a critical interrupt request occurs, SCVCR can be read. If there are multiple critical interrupt sources, SCVCR latches the highest priority critical interrupt. Note that CVECx field will correctly reflect only first 64 interrupt vectors (See Table 8-6 for details). The value of SCVEC cannot change while it is being read.
6–24	—	Write ignored, read = 0
25–31	CVEC	Critical interrupt vector. Specifies a 7-bit unique number of the IPIC’s highest priority critical interrupt source, pending to the core. When a critical interrupt request occurs, SCVCR can be read. If there are multiple critical interrupt sources, SCVCR latches the highest priority critical interrupt. Note that CVEC field correctly reflects all of the interrupt vectors (See Table 8-6 for details). The value of SCVEC cannot change while it is being read.

### 8.5.23 System Management Interrupt Vector Register (SMVCR)

SMVCR, shown in Figure 8-27, contains a 7-bit code (Table 8-33) representing the unmasked system management interrupt (*smi*) source of the highest priority level.

Note that in core disabled mode, the user should use SIVCR only read an updated interrupt vector (SMVCR should not be used).



**Figure 8-27. System Management Interrupt Vector Register (SMVCR)**

Table 8-33 defines the bit fields of SMVCR.



Table 8-33. SMVCR Field Descriptions

Bits	Name	Description
0–5	MVECx	Backward (MPC8260) compatible system management interrupt vector. Specifies a 6-bit unique number of the IPIC's highest priority system management interrupt source, pending to the core. When a system management interrupt request occurs, SMVCR can be read. If there are multiple system management interrupt sources, SMVCR latches the highest priority system management interrupt. Note that MVECx f correctly reflects only the first 64 interrupt vectors (See Table 8-6 for details). The value of SMVEC cannot change while it is being read.
6–24	—	Write ignored, read = 0
25–31	MVEC	System management interrupt vector. Specifies a 7-bit unique number of the IPIC's highest priority system management interrupt source, pending to the core. When a system management interrupt request occurs, SMVCR can be read. If there are multiple system management interrupt sources, SMVCR latches the highest priority system management interrupt. Note that MVEC field will correctly reflect all interrupt vectors (See Table 8-6 for details). The value of SMVEC cannot change while it is being read.

## 8.6 Functional Description

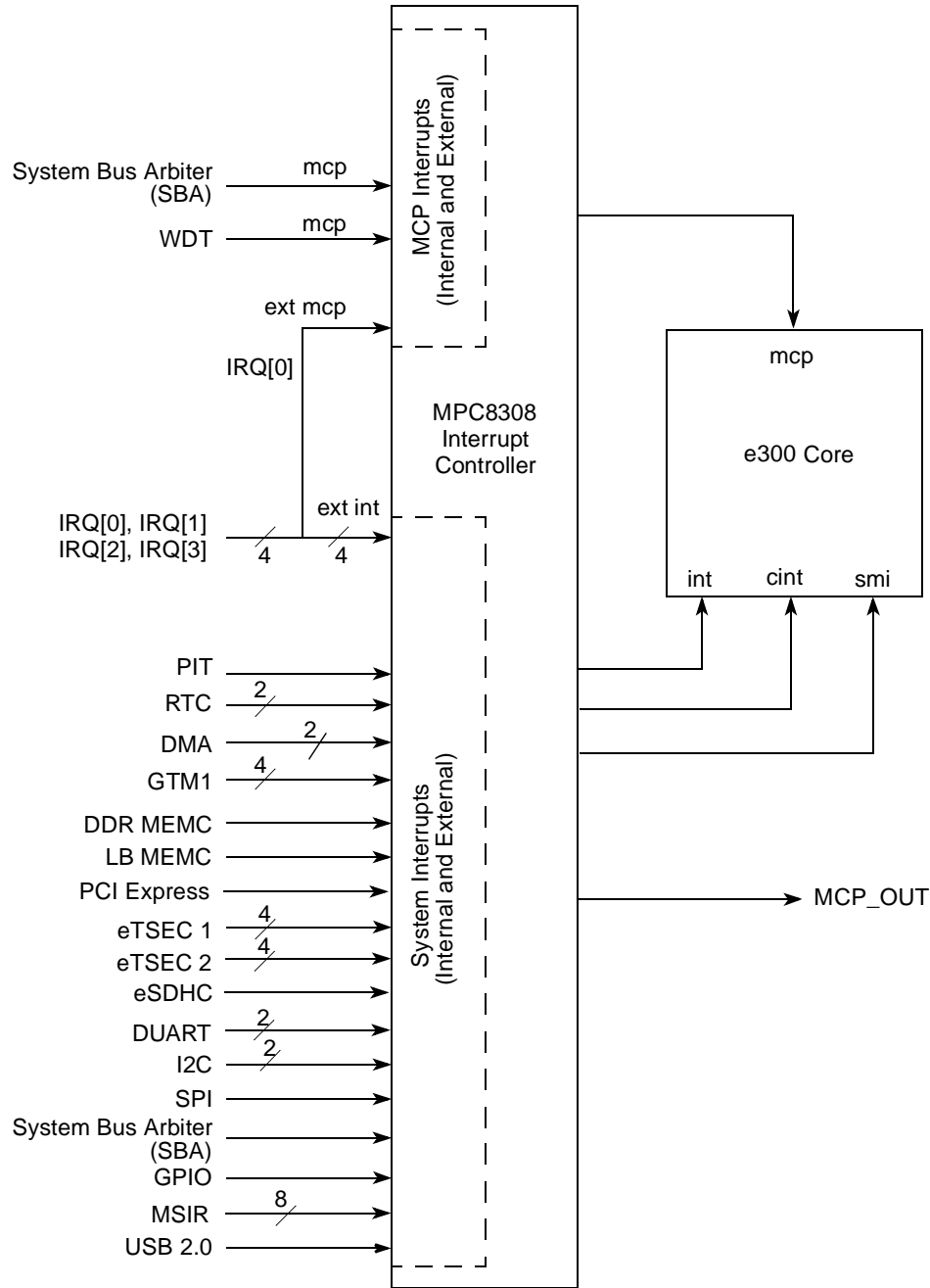
The following sections describe the types of interrupts, interrupt configurations, and their priorities.

### 8.6.1 Interrupt Types

The IPIC is responsible for receiving hardware-generated interrupts from different sources (both internal and external) along with prioritizing and delivering them to the CPU for servicing. The interrupt sources are controlled by the IPIC unit and may cause three types of exceptions in the processor core. The  $\overline{int}$  signal is the main interrupt output from the IPIC to the processor core and causes the external interrupt exception. The  $\overline{cint}$  signal is the critical interrupt output from the IPIC to the processor core and causes the critical external interrupt exception. The  $\overline{smi}$  signal is the system management interrupt output from the IPIC to the processor core and causes the system management interrupt exception. The machine check exception is caused by the internal  $\overline{mcp}$  signal generated by the IPIC, informing the processor of error conditions, assertion of the external MCP request, and other conditions.

## 8.6.2 Interrupt Configuration

Figure 8-28 shows the interrupt configuration.



**Figure 8-28. Interrupt Structure**

The interrupt controller allows masking of each interrupt source. When an unmasked interrupt source is pending in the SIPNR register, the interrupt controller sends an interrupt request to the core. When an interrupt is taken, the interrupt mask bit in the machine state register is cleared to disable further interrupt requests to the e300 core until software can handle them.

All interrupt sources are prioritized and bits are set in the system interrupt pending register (SIPNR, SEPNR) as interrupts occur regardless of whether they are masked in the IPIC. The prioritization of the interrupt sources is flexible within the following groups:

- The relative priority of the eTSEC1 Tx, eTSEC1 Rx, eTSEC1 Err, eTSEC2 Tx, eTSEC2 Rx, eTSEC2 Err, and USB DR internal interrupt signals can be modified.
- The relative priority of the eSDHC internal interrupt signal can be modified.
- The relative priority of the PCI Express, DMAC, and MSIR1 internal interrupts can be modified.
- The relative priority of the UART1, UART2, eTSEC1 1588, eTSEC2 1588, I2C1, I2C2, and SPI internal interrupt signals can be modified.
- The relative priority of the RTC SEC, PIT, and MSIR0 internal interrupts along with IRQ0, IRQ1, IRQ2, and IRQ3 external interrupts can be modified.
- The relative priority of the RTC ALR and SBA internal interrupts can be modified.
- One interrupt source can be assigned to be the programmable highest priority.

The SIVCR is updated with a 7-bit vector corresponding to the sub-block with the highest current priority.

### 8.6.3 Internal Interrupts Group Relative Priority

The relative priority in each internal group is programmable and can be changed dynamically. The group priorities are programmed in the IPIC internal interrupt priority registers (SIPRR<sub>x</sub>) and can be changed dynamically to implement a rotating priority.

In addition, the grouping of the locations of the interrupt entries has the following two options:

- Grouped.  
In the group scheme, all interrupts are grouped together at the top of [Table 8-34](#), ahead of most other interrupt sources. This scheme is ideal for applications where all interrupt sources function at a very high data rate and interrupt latency is very important.
- Spread.  
In the spread scheme, priorities are spread over [Table 8-34](#) so other sources can have lower interrupt latencies. This scheme is also programmed but cannot be changed dynamically.

### 8.6.4 Mixed Interrupts Group Relative Priority

The relative priority between up to four internal and four external interrupts in each group is programmable and can be changed dynamically. The group priorities are programmed in the IPIC mixed interrupt priority registers (SMPRR<sub>x</sub>) and can be changed dynamically to implement a rotating priority.

In addition, the grouping of the locations of the mixed interrupt entries has the following two options:

- Grouped.  
In the group scheme, all interrupts are grouped together at the top of the priority table, ahead of most other interrupt sources. For more information, see [Table 8-34](#). This scheme is ideal for applications where all interrupt sources function at a very high data rate and interrupt latency is very important.

- Spread.

In the spread scheme, priorities are spread over the table so other sources can have lower interrupt latencies. This scheme is also programmed but cannot be changed dynamically.

### 8.6.5 Highest Priority Interrupt

In addition to the group relative priority option, SICFR[HPI] can be used to specify one interrupt source as having the highest priority. This interrupt remains within the same interrupt level as the other interrupt controller interrupts, but is serviced before any other interrupt in [Table 8-34](#).

If the highest priority feature is not used, the IPIC selects the interrupt request in MIXA0 to be the highest priority interrupt and the standard interrupt priority order is used from [Table 8-34](#). SICFR[HPI] can be updated dynamically to allow the user to change a normally low priority source into a high priority-source for a period as needed.

### 8.6.6 Interrupt Source Priorities

Each of the IPIC's internal and external interrupt sources can independently assert one interrupt request to the core. [Table 8-34](#) shows the prioritization of these interrupt sources. As described in previous sections, flexibility exists in the relative ordering of the interrupts, but, in general, relative priorities are as shown. A single interrupt priority number is associated with each table entry.

**Table 8-34. Interrupt Source Priority Levels**

Priority	Interrupt Source Description
1	Highest
2	MIXA0 (Grouped/Spread)
3	MIXA1 (Grouped)
4	MIXA2 (Grouped)
5	MIXA3 (Grouped)
6	MIXB0 (Spread)
7	SYSB0 (Grouped)
8	SYSB1 (Grouped)
9	SYSB2 (Grouped)
10	SYSB3 (Grouped)
11	MIXA1 (Spread)
12	SYSB4 (Grouped)
13	SYSB5 (Grouped)
14	SYSB6 (Grouped)
15	SYSB7 (Grouped)
16	MIXB0 (Grouped)
17	MIXB1 (Grouped)
18	MIXB2 (Grouped)
19	MIXB3 (Grouped)

Table 8-34. Interrupt Source Priority Levels (continued)

Priority	Interrupt Source Description
20	MIXB1 (Spread)
21	SYSA0 (Grouped)
22	SYSA1 (Grouped)
23	SYSA2 (Grouped)
24	SYSA3 (Grouped)
25	MIXA2 (Spread)
26	SYSA4 (Grouped)
27	SYSA5 (Grouped)
28	SYSA6 (Grouped)
29	SYSA7 (Grouped)
30	MIXA4 (Grouped)
31	MIXA5 (Grouped)
32	MIXA6 (Grouped)
33	MIXA7 (Grouped)
34	MIXB2 (Spread)
35	SYSC0 (Grouped)
36	SYSC1 (Grouped)
37	SYSC2 (Grouped)
38	SYSC3 (Grouped)
39	MIXA3 (Spread)
40	SYSC4 (Grouped)
41	SYSC5 (Grouped)
42	SYSC6 (Grouped)
43	SYSC7 (Grouped)
44	MIXB4 (Grouped)
45	MIXB5 (Grouped)
46	MIXB6 (Grouped)
47	MIXB7 (Grouped)
48	MIXB3 (Spread)
49	SYSD0 (Grouped)
50	SYSD1 (Grouped)
51	SYSD2 (Grouped)
52	SYSD3 (Grouped)

**Table 8-34. Interrupt Source Priority Levels (continued)**

Priority	Interrupt Source Description
53	MIXA4 (Spread)
54	SYSD4 (Grouped)
55	SYSD5 (Grouped)
56	SYSD6 (Grouped)
57	SYSD7 (Grouped)
58	MIXB4 (Spread)
59	GTM1_4
60	SYSB0 (Spread)
61	SYSA0 (Spread)
62	Reserved
63	SYSC0 (Spread)
64	SYSD0 (Spread)
65	Reserved
66	GPIO
67	MIXA5 (Spread)
68	Reserved
69	SYSB1 (Spread)
70	SYSA1 (Spread)
71	DDRC
72	SYSC1 (Spread)
73	SYSD1 (Spread)
74	Reserved
75	LBC
76	MIXB5 (Spread)
77	GTM1_2
78	SYSB2 (Spread)
79	SYSA2 (Spread)
80	Reserved
81	SYSC2 (Spread)
82	SYSD2 (Spread)
83	Reserved
84	Reserved
85	MIXA6 (Spread)
86	MSIR2
87	SYSB3 (Spread)
88	SYSA3 (Spread)

Table 8-34. Interrupt Source Priority Levels (continued)

Priority	Interrupt Source Description
89	MSIR3
90	SYSC3 (Spread)
91	SYSD3 (Spread)
92	Reserved
93	Reserved
94	MIXB6 (Spread)
95	GTM1_3
96	SYSB4 (Spread)
97	SYSA4 (Spread)
98	Reserved
99	SYSC4 (Spread)
100	SYSD4 (Spread)
101	Reserved
102	MSIR4
103	MIXA7 (Spread)
104	MSIR5
105	SYSB5 (Spread)
106	SYSA5 (Spread)
107	MSIR6
108	SYSC5 (Spread)
109	SYSD5 (Spread)
110	Reserved
111	MSIR7
112	MIXB7 (Spread)
113	GTM1_1
114	SYSB6 (Spread)
115	SYSA6 (Spread)
116	Reserved
117	SYSC6 (Spread)
118	SYSD6 (Spread)
119	Reserved
120	Reserved
121	Reserved
122	SYSB7 (Spread)
123	SYSA7 (Spread)

**Table 8-34. Interrupt Source Priority Levels (continued)**

Priority	Interrupt Source Description
124	DMAC Err
125	SYSC7 (Spread)
126	SYSD7 (Spread)
127	Reserved
128	Reserved

### 8.6.7 Masking Interrupt Sources

By programming the system interrupt mask registers,  $SIMSR_x$  and  $SEMSR$ , the user can mask interrupt requests to the core. Each  $SIMSR_x$  and  $SEMSR$  bit corresponds to an interrupt source. To enable an interrupt, set the corresponding  $SIMSR$  or  $SEMSR$  bit. When a masked interrupt source has a pending interrupt request, the corresponding  $SIPNR_x$  or  $SEMSR$  bit is set, even though the interrupt is not generated to the core. The user can mask all interrupt sources to implement a polling interrupt servicing scheme.

When an interrupt source has multiple interrupting events, the user can individually mask these events by programming a mask register within that particular block. [Table 8-34](#) shows the interrupt sources that have multiple interrupting events.



Figure 8-29 shows an example of how the masking occurs using a DDR block.

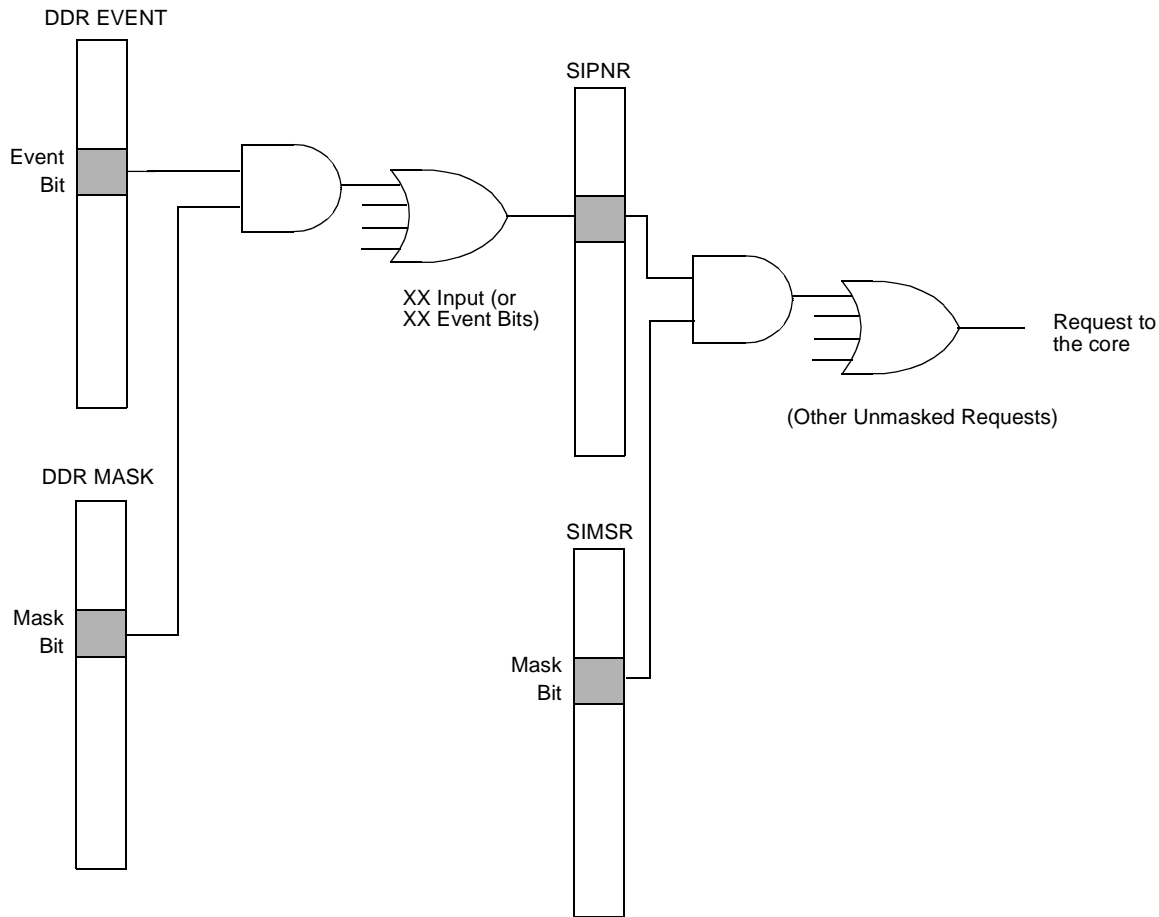


Figure 8-29. DDR Interrupt Request Masking

### 8.6.8 Interrupt Vector Generation and Calculation

Pending unmasked interrupts are presented to the core in order of priority according to Table 8-34. The interrupt vector that allows the core to locate the interrupt service routine is made available to the core by interrupt handler software reading SIVCR. The interrupt controller passes an interrupt vector corresponding to the highest-priority, unmasked, pending interrupt in response to a read of SIVCR.

Table 8-5 lists the encodings for the seven low-order bits of the interrupt vector.

### 8.6.9 Machine Check Interrupts

The IPIC supports the non-maskable machine check interrupts. When an error interrupt signal is received, the interrupt controller indicates the source by setting the corresponding SERSR bit. These sources are listed in Table 8-23.

## 8.7 Message Shared Interrupts

The message shared interrupt (MSI) registers enable the PCI Express end points to generate interrupt requests to the local e300 CPU. Each end point can generate an interrupt and set a unique bit in one of the eight MSIR registers. Clearing the MSIR register happens immediately after the read of its content, and by then a new set operation can begin.  $MSIR_n$  is considered active if it contains at least one bit set. Each active non masked  $MSIR_n$  register will generate an interrupt.

### 8.7.1 Memory Map/Register Definition

The MSI programmable register map occupies 64 bytes of memory-mapped space. The MSI registers are 32-bit wide and must be accessed in a 32-bit read or write operation. The listed addresses are offset from the IPIC base address. [Table 8-35](#) shows the message shared registers address map.

**Table 8-35. Message Shared Registers Address Map**

Offset	Register	Access	Reset	Section/page
0xC0	MSIR0—Message shared interrupt register 0	Special	0x0000_0000	<a href="#">8.7.2.1/8-40</a>
0xC4	MSIR1—Message shared interrupt register 1	Special	0x0000_0000	<a href="#">8.7.2.1/8-40</a>
0xC8	MSIR2—Message shared interrupt register 2	Special	0x0000_0000	<a href="#">8.7.2.1/8-40</a>
0xCC	MSIR3—Message shared interrupt register 3	Special	0x0000_0000	<a href="#">8.7.2.1/8-40</a>
0xD0	MSIR4—Message shared interrupt register 4	Special	0x0000_0000	<a href="#">8.7.2.1/8-40</a>
0xD4	MSIR5—Message shared interrupt register 5	Special	0x0000_0000	<a href="#">8.7.2.1/8-40</a>
0xD8	MSIR6—Message shared interrupt register 6	Special	0x0000_0000	<a href="#">8.7.2.1/8-40</a>
0xDC	MSIR7—Message shared interrupt register 7	Special	0x0000_0000	<a href="#">8.7.2.1/8-40</a>
0xE0–0xEC	Reserved	—	—	—
0xF0	MSIMR—Message shared interrupt mask register	R/W	0x0000_0000	<a href="#">8.7.2.2/8-41</a>
0xF4	MSISR—Message shared interrupt status register	R	0x0000_0000	<a href="#">8.7.2.3/8-42</a>
0xF8	MSIIR—Message shared interrupt index register	W	0x0000_0000	<a href="#">8.7.2.4/8-42</a>
0xFC	Reserved	—	—	—

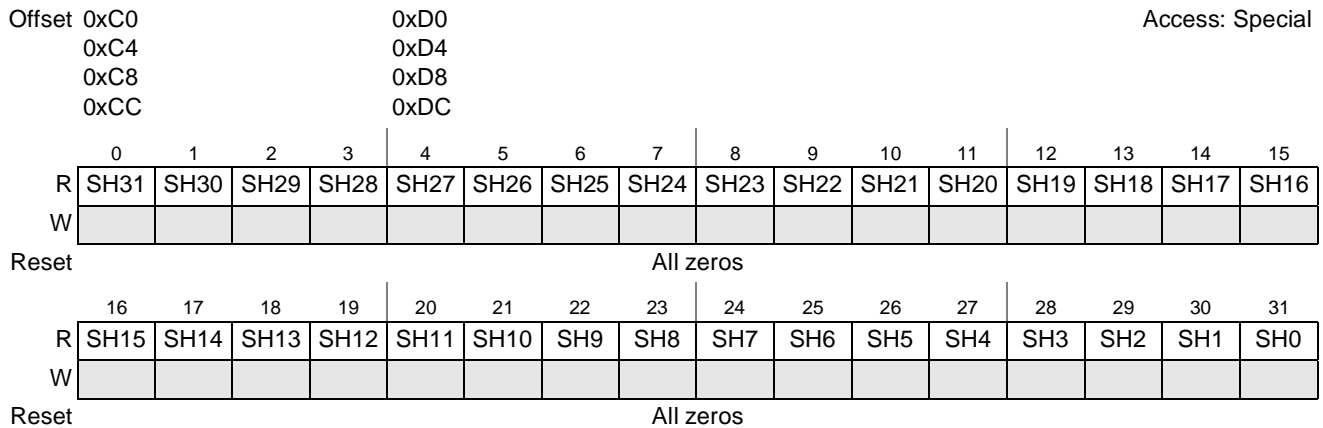
### 8.7.2 Message Shared Registers

This section contains the description of all of the message shared interrupt registers.

#### 8.7.2.1 Message Shared Interrupt Register (MSIRs)

There are eight MSIRs indicating the interrupt sources that share the message have pending interrupts. Up to 32 sources can share any individual message register. These registers are cleared when read. A write to these registers has no effect.

Figure 8-30 shows the message shared interrupt registers.



**Figure 8-30. Message Shared Interrupt Register (MSIRs)**

Table 8-36 describes the bits of the MSIRs.

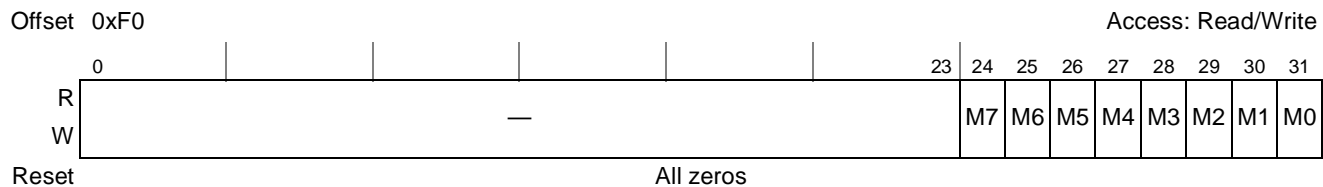
**Table 8-36. MSIRs Field Descriptions**

Bits	Name	Description
31–0	$SH_n$	Message sharer $n$ ( $n = 31-0$ ) has a pending interrupt.

### 8.7.2.2 Message Shared Interrupt Mask Register (MSIMR)

The MSIMR contains the mask bits for the message shared interrupt register interrupts. The mask bit corresponding to a message shared interrupt register must be clear to enable interrupt generation when the message input region is written and a bit in the message shared interrupt register is set. This is a read-write register.

Figure 8-31 shows the message shared interrupt mask register.



**Figure 8-31. Message Shared Interrupt Mask Register (MSIMR)**

Table 8-37 describes the bits of the MSIMR.

**Table 8-37. MSIMR Field Descriptions**

Bits	Name	Description
0–23	—	Reserved.
24	M7	Mask 7. Set to 1 masks interrupt generation for message shared interrupt register 7
25	M6	Mask 6. Set to 1 masks interrupt generation for message shared interrupt register 6

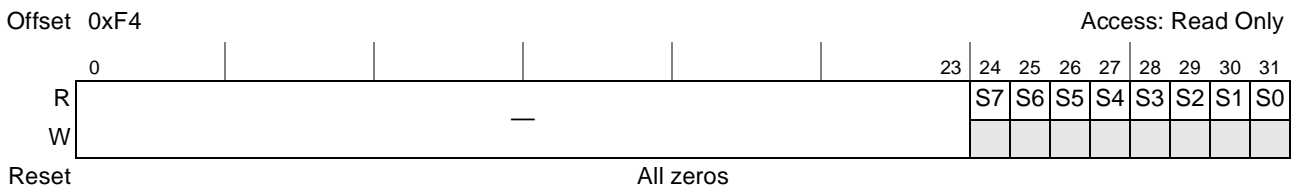
**Table 8-37. MSIMR Field Descriptions (continued)**

Bits	Name	Description
26	M5	Mask 5. Set to 1 masks interrupt generation for message shared interrupt register 5
27	M4	Mask 4. Set to 1 masks interrupt generation for message shared interrupt register 4
28	M3	Mask 3. Set to 1 masks interrupt generation for message shared interrupt register 3
29	M2	Mask 2. Set to 1 masks interrupt generation for message shared interrupt register 2
30	M1	Mask 1. Set to 1 masks interrupt generation for message shared interrupt register 1
31	M0	Mask 0. Set to 1 masks interrupt generation for message shared interrupt register 0

### 8.7.2.3 Message Shared Interrupt Status Register (MSISR)

This register contains the status bits for the message shared interrupts. The status bit is set to 1 when the corresponding message shared interrupt is active. The status bit is 0 if all the corresponding shared interrupt sources are cleared in the message shared interrupt register (MSIR). This register is read-only.

Figure 8-32 shows the message shared interrupt status register.



**Figure 8-32. Message Shared Interrupt Status Register (MSISR)**

Table 8-38 describes the bits of the MSISR.

**Table 8-38. MSISR Field Descriptions**

Bits	Name	Description
0–23	—	Reserved.
24	S7	Status 7. Set to 1 when message shared interrupt 7 is active
25	S6	Status 6. Set to 1 when message shared interrupt 6 is active
26	S5	Status 5. Set to 1 when message shared interrupt 5 is active
27	S4	Status 4. Set to 1 when message shared interrupt 4 is active
28	S3	Status 3. Set to 1 when message shared interrupt 3 is active
29	S2	Status 2. Set to 1 when message shared interrupt 2 is active
30	S1	Status 1. Set to 1 when message shared interrupt 1 is active
31	S0	Status 0. Set to 1 when message shared interrupt 0 is active

### 8.7.2.4 Message Shared Interrupt Index Register (MSIIR)

This register provides a mechanism for setting an interrupt in the message shared interrupt registers. There are two fields. When this register is written, one field selects the register in which an interrupt bit is to be set and the other field selects the bit in the selected register to set. This register is write-only.

Figure 8-33 shows the message shared interrupt index register.



**Figure 8-33. Message Shared Interrupt Index Register (MSIIR)**

Table 8-39 describes the bits of the MSIIRs.

**Table 8-39. MSIIR Field Descriptions**

Bits	Name	Description
0–2	SRS	Shared interrupt register select. Select the message shared interrupt register. 000 Message shared interrupt register 0 001 Message shared interrupt register 1 010 Message shared interrupt register 2 ... 111 Message shared interrupt register 7
3–7	IBS	Interrupt bit select. Select the bit to set. 00000 Set bit 31 (SH0) 00001 Set bit 30 (SH1) 00010 Set bit 29 (SH2) ... 11111 Set bit 0 (SH31)
8–31	—	Reserved



# Chapter 9

## DDR Memory Controller

### 9.1 Introduction

The fully programmable DDR SDRAM controller supports most JEDEC standard  $\times 8$ ,  $\times 16$ , and  $\times 32$  DDR2 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.
- MPC8308 supports only DDR2 controller. In this chapter, usage of the word ‘DDR’ is generic and refers to the DDR2 controller.

Figure 9-1 is a high-level block diagram of the DDR memory controller with its associated interfaces. Section 9.5, “Functional Description,” contains detailed figures of the controller.

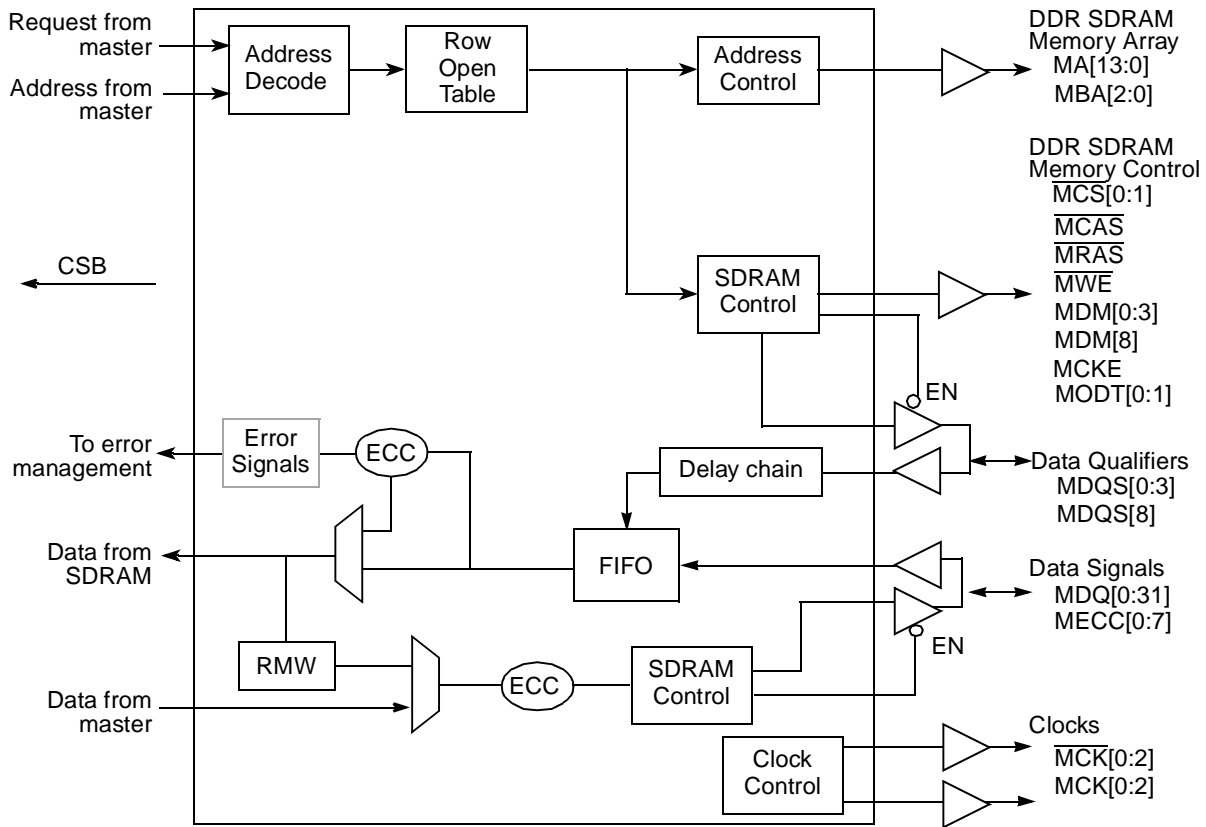


Figure 9-1. DDR Memory Controller Simplified Block Diagram

## 9.2 Features

The DDR memory controller includes these distinctive features:

- Support for DDR2 SDRAM
- Supports 8-bit ECC
- Supports 16-/32-bit data interface
- Programmable settings for meeting all SDRAM timing parameters
- The following SDRAM configurations are supported:
  - As many as two physical banks (chip selects), each bank independently addressable upto 512 Mbytes
    - 64-Mbit to 2-Gbit devices depending on internal device configuration with  $\times 8/\times 16/\times 32$  data ports (no direct  $\times 4$  support)
    - One 32-bit device, one 16-bit device or two 8-bit devices on a 16-bit bus, or two 16-bit devices or four 8-bit devices on a 32-bit bus



- Unbuffered and registered DIMMs
- Chip select interleaving 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)
- Open page management (dedicated entry for each logical bank)
- Automatic DRAM initialization sequence or software-controlled initialization sequence
- Automatic DRAM data initialization
- Support for up to eight posted refreshes
- Memory controller clock frequency of two times the SDRAM clock with support for sleep power management
- Support for error injection

### 9.2.1 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<sub>n</sub>\_CONFIG[AP<sub>n</sub>\_EN].

## 9.3 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.

### 9.3.1 Signals Overview

Memory controller signals are grouped as follows:

- Memory interface signals
- Clock signals
- Debug signals

Table 9-1 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 9-1. DDR Memory Interface Signal Summary**

Name	Function/Description	Reset	Pins	I/O
MDQ[0:31]	Data bus	All zeros	32	I/O
MDQS[0:3]	Data strobes	All zeros	4	I/O
MDQS[8]	Data strobe for MECC[0:7]	All zeros	1	I/O
MECC[0:7]	Error checking and correcting	All zeros	8	I/O
$\overline{\text{MCAS}}$	Column address strobe	One	1	O
MA[13:0]	Address bus	All zeros	14	O
MBA[2:0]	Logical bank address	All zeros	3	O
$\overline{\text{MCS}}$ [0:1]	Chip selects	All ones	2	O
$\overline{\text{MWE}}$	Write enable	One	1	O
$\overline{\text{MRAS}}$	Row address strobe	One	1	O
MDM[0:3]	Data mask	All zeros	4	O
MDM[8]	Data mask for MECC[0:7]	All zeros	1	O
MCK[0:2]	DRAM clock outputs	Zero	3	O
$\overline{\text{MCK}}$ [0:2]	DRAM clock outputs (complement)	One	3	O
MCKE	DRAM clock enable	Zero	1	O
MODT[0:1]	DRAM on-die termination external control.	All zeros	2	O
$\text{MV}_{\text{REF}}$	DDR2 DRAM reference	—	1	I

Table 9-2 shows the memory address signal mappings.

**Table 9-2. Memory Address Signal Mappings**

Signal Name (Outputs)		JEDEC DDR DIMM Signals (Inputs)
msb	MA13	A13
	MA12	A12
	MA11	A11
	MA10	A10 (AP for DDR) <sup>1</sup>
	MA9	A9
	MA8	A8 (alternate AP for DDR) <sup>2</sup>
	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

<sup>1</sup> Auto-precharge for DDR signaled on A10 when DDR\_SDRAM\_CFG[PCHB8] = 0

<sup>2</sup> Auto-precharge for DDR signaled on A8 when DDR\_SDRAM\_CFG[PCHB8] = 1

## 9.3.2 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.

### 9.3.2.1 Memory Interface Signals

Table 9-3 describes the DDR controller memory interface signals.

**Table 9-3. Memory Interface Signals—Detailed Signal Descriptions**

Signal	I/O	Description	
MDQ[0:31]	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.	
		<b>State Meaning</b>	Asserted/Negated—Represent the value of data being driven by the DDR memory controller.
		<b>Timing</b>	Assertion/Negation—Driven coincident with corresponding data strobes (MDQS) signal. 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.	
		<b>State Meaning</b>	Asserted/Negated—Represents the state of data being driven by the external DDR SDRAMs.
<b>Timing</b>		Assertion/Negation—The DDR SDRAM drives data during a READ transaction. High impedance—No READ or WRITE command in progress; data is not being driven by the memory controller or the DRAM.	
MDQS[0:3], MDQS[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.
			<b>State Meaning</b>
	<b>Timing</b>	Assertion/Negation—If a WRITE command is registered at clock edge, data strobes at the DRAM assert centered in the data eye. For more information, see the JEDEC DDR2 SDRAM specification.	
	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.	
		<b>State Meaning</b>	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 9-36 for byte lane assignments.
<b>Timing</b>		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 9-3. 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	ECC signals represent the state of ECC driven by the DDR controller on writes. See <a href="#">Section 9.5.11, "Error Checking and Correcting (ECC),"</a> for more details.	
		<b>State Meaning</b>	Asserted/Negated—Represents the state of ECC being driven by the DDR controller on writes.
		<b>Timing</b>	Assertion/Negation—Same timing as MDQ High impedance—Same timing as MDQ
	I	As inputs, the ECC signals represent the state of ECC driven by the SDRAM devices on reads.	
		<b>State Meaning</b>	Asserted/Negated—Represents the state of ECC being driven by the DDR SDRAMs on reads.
<b>Timing</b>		Assertion/Negation—Same timing as MDQ High impedance—Same timing as MDQ	
MA[13:0]	O	Address bus. Memory controller outputs for the address to the DRAM. MA[13:0] carry 14 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.	
		<b>State Meaning</b>	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 <a href="#">Table 9-38</a> for a complete description of the mapping of these signals.
		<b>Timing</b>	Assertion/Negation—The address is always driven when the memory controller is enabled. It is valid when a transaction is driven to DRAM (when $\overline{MCS}_n$ is active). 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.	
		<b>State Meaning</b>	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. <a href="#">Table 9-38</a> describes the mapping of these signals in all cases.
		<b>Timing</b>	Assertion/Negation—Same timing as $MA_n$ High impedance—Same timing as $MA_n$
$\overline{MCAS}$	O	Column address strobe. Active-low SDRAM address multiplexing signal. $\overline{MCAS}$ is asserted for read or write transactions and for mode register set, refresh, and precharge commands.	
		<b>State Meaning</b>	Asserted—Indicates that a valid SDRAM column address is on the address bus for read and write transactions. See <a href="#">Table 9-41</a> for more information on the states required on $\overline{MCAS}$ for various other SDRAM commands. Negated—The column address is not guaranteed to be valid.
		<b>Timing</b>	Assertion/Negation—Assertion and negation timing is directed by the values described in <a href="#">Section 9.4.1.4, "DDR SDRAM Timing Configuration 0 (TIMING_CFG_0),"</a> <a href="#">Section 9.4.1.5, "DDR SDRAM Timing Configuration 1 (TIMING_CFG_1),"</a> <a href="#">Section 9.4.1.6, "DDR SDRAM Timing Configuration 2 (TIMING_CFG_2),"</a> and <a href="#">Section 9.4.1.3, "DDR SDRAM Timing Configuration 3 (TIMING_CFG_3)." </a> High impedance— $\overline{MCAS}$ is always driven unless the memory controller is disabled.

**Table 9-3. Memory Interface Signals—Detailed Signal Descriptions (continued)**

Signal	I/O	Description
$\overline{\text{MRAS}}$	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.
		<b>State Meaning</b>
		<b>Timing</b>
$\overline{\text{MCS}}[0:1]$	O	Chip selects. Two chip selects supported by the memory controller.
		<b>State Meaning</b>
		<b>Timing</b>
$\overline{\text{MWE}}$	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.
		<b>State Meaning</b>
		<b>Timing</b>
MDM[0:3], MDM[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 MDM3 corresponds to the LSB, while MDM4 corresponds to the ECC byte. Table 9-36 shows byte lane encodings.
		<b>State Meaning</b>
		<b>Timing</b>
MODT[0:1]	O	On-Die termination. Memory controller outputs for the ODT to the DRAM. MODT[0:1] represents the on-die termination for the associated data, data masks, ECC, and data strobes.
		<b>State Meaning</b>
		<b>Timing</b>

### 9.3.2.2 Clock Interface Signals

Table 9-4 contains the detailed descriptions of the clock signals of the DDR controller.

**Table 9-4. Clock Signals—Detailed Signal Descriptions**

Signal	I/O	Description
MCK[0:2], $\overline{\text{MCK}}[0:2]$	O	DRAM clock output and its complement. See <a href="#">Section 9.5.4.1, “Clock Distribution.”</a>
		<b>State Meaning</b> 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.
		<b>Timing</b> Assertion/Negation—Timing is controlled by the DDR_CLK_CNTL register at offset 0x130.
MCKE	O	Clock enable. Output signals used as the clock enables to the SDRAM. MCKE can be negated to stop clocking the DDR SDRAM.
		<b>State Meaning</b> Asserted—Clocking to the SDRAM is enabled. 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 is negated.
		<b>Timing</b> 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. High impedance—Always driven.

## 9.4 Memory Map/Register Definition

Table 9-5 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 9-5. DDR Memory Controller Memory Map**

Offset	Register	Access	Reset	Section/Page
<b>DDR Memory Controller—Block Base Address 0x0_2000</b>				
0x000	CS0_BNDS—Chip select 0 memory bounds	R/W	0x0000_0000	<a href="#">9.4.1.1/9-10</a>
0x008	CS1_BNDS—Chip select 1 memory bounds	R/W	0x0000_0000	<a href="#">9.4.1.1/9-10</a>
0x080	CS0_CONFIG—Chip select 0 configuration	R/W	0x0000_0000	<a href="#">9.4.1.2/9-11</a>
0x084	CS1_CONFIG—Chip select 1 configuration	R/W	0x0000_0000	<a href="#">9.4.1.2/9-11</a>
0x100	TIMING_CFG_3—DDR SDRAM timing configuration 3	R/W	0x0000_0000	<a href="#">9.4.1.3/9-13</a>
0x104	TIMING_CFG_0—DDR SDRAM timing configuration 0	R/W	0x0011_0105	<a href="#">9.4.1.4/9-14</a>
0x108	TIMING_CFG_1—DDR SDRAM timing configuration 1	R/W	0x0000_0000	<a href="#">9.4.1.5/9-16</a>
0x10C	TIMING_CFG_2—DDR SDRAM timing configuration 2	R/W	0x0000_0000	<a href="#">9.4.1.6/9-18</a>

Table 9-5. DDR Memory Controller Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
0x110	DDR_SDRAM_CFG—DDR SDRAM control configuration	R/W	0x0200_0000	9.4.1.7/9-19
0x114	DDR_SDRAM_CFG_2—DDR SDRAM control configuration 2	R/W	0x0000_0000	9.4.1.8/9-22
0x118	DDR_SDRAM_MODE—DDR SDRAM mode configuration	R/W	0x0000_0000	9.4.1.9/9-23
0x11C	DDR_SDRAM_MODE_2—DDR SDRAM mode configuration 2	R/W	0x0000_0000	9.4.1.10/9-24
0x120	DDR_SDRAM_MD_CNTL—DDR SDRAM mode control	R/W	0x0000_0000	9.4.1.11/9-25
0x124	DDR_SDRAM_INTERVAL—DDR SDRAM interval configuration	R/W	0x0000_0000	9.4.1.12/9-27
0x128	DDR_DATA_INIT—DDR SDRAM data initialization	R/W	0x0000_0000	9.4.1.13/9-28
0x130	DDR_SDRAM_CLK_CNTL—DDR SDRAM clock control	R/W	0x0200_0000	9.4.1.14/9-28
0x140– 0x144	Reserved	—	—	—
0x148	DDR_INIT_ADDR—DDR training initialization address	R/W	0x0000_0000	9.4.1.15/9-29
0x150– 0xBF4	Reserved	—	—	—
0xBF8	DDR_IP_REV1—DDR IP block revision 1	R	0xn <sub>nnnn</sub> _n <sub>nnn</sub> <sup>1</sup>	9.4.1.16/9-29
0xBFC	DDR_IP_REV2—DDR IP block revision 2	R	0x00n <sub>n</sub> _00n <sub>n</sub> <sup>1</sup>	9.4.1.17/9-30
0xE00	DATA_ERR_INJECT_HI—Memory data path error injection mask high	R/W	0x0000_0000	9.4.1.18/9-30
0xE04	DATA_ERR_INJECT_LO—Memory data path error injection mask low	R/W	0x0000_0000	9.4.1.19/9-31
0xE08	ERR_INJECT—Memory data path error injection mask ECC	R/W	0x0000_0000	9.4.1.20/9-31
0xE20	CAPTURE_DATA_HI—Memory data path read capture high	R/W	0x0000_0000	9.4.1.21/9-32
0xE24	CAPTURE_DATA_LO—Memory data path read capture low	R/W	0x0000_0000	9.4.1.22/9-32
0xE28	CAPTURE_ECC—Memory data path read capture ECC	R/W	0x0000_0000	9.4.1.23/9-33
0xE40	ERR_DETECT—Memory error detect	w1c	0x0000_0000	9.4.1.24/9-33
0xE44	ERR_DISABLE—Memory error disable	R/W	0x0000_0000	9.4.1.25/9-34
0xE48	ERR_INT_EN—Memory error interrupt enable	R/W	0x0000_0000	9.4.1.26/9-35
0xE4C	CAPTURE_ATTRIBUTES—Memory error attributes capture	R/W	0x0000_0000	9.4.1.27/9-36
0xE50	CAPTURE_ADDRESS—Memory error address capture	R/W	0x0000_0000	9.4.1.28/9-37
0xE54	Reserved	—	—	—
0xE58	ERR_SBE—Single-Bit ECC memory error management	R/W	0x0000_0000	9.4.1.29/9-37

<sup>1</sup> Implementation-dependent reset values are listed in specified section/page.

## 9.4.1 Register Descriptions

This section describes the DDR memory controller registers. Shading indicates reserved fields that should not be written.

### 9.4.1.1 Chip Select Memory Bounds (CS<sub>n</sub>\_BNDS)

The chip select bounds registers (CS<sub>n</sub>\_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<sub>n</sub>\_BNDS should equal the size of physical DRAM. Also, note that EA<sub>n</sub> must be greater than or equal to SA<sub>n</sub>.



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<sub>*n*</sub>\_BNDS are shown in Figure 9-2.



Figure 9-2. Chip Select Bounds Registers (CS<sub>*n*</sub>\_BNDS)

Table 9-6 describes the CS<sub>*n*</sub>\_BNDS register fields.

Table 9-6. CS<sub>*n*</sub>\_BNDS Field Descriptions

Bits	Name	Description
0–7	—	Reserved
8–15	SAn	Starting address for chip select (bank) <i>n</i> . This value is compared against the 8 msbs of the 32-bit address.
16–23	—	Reserved
24–31	EAn	Ending address for chip select (bank) <i>n</i> . This value is compared against the 8 msbs of the 32-bit address.

### 9.4.1.2 Chip Select Configuration (CS<sub>*n*</sub>\_CONFIG)

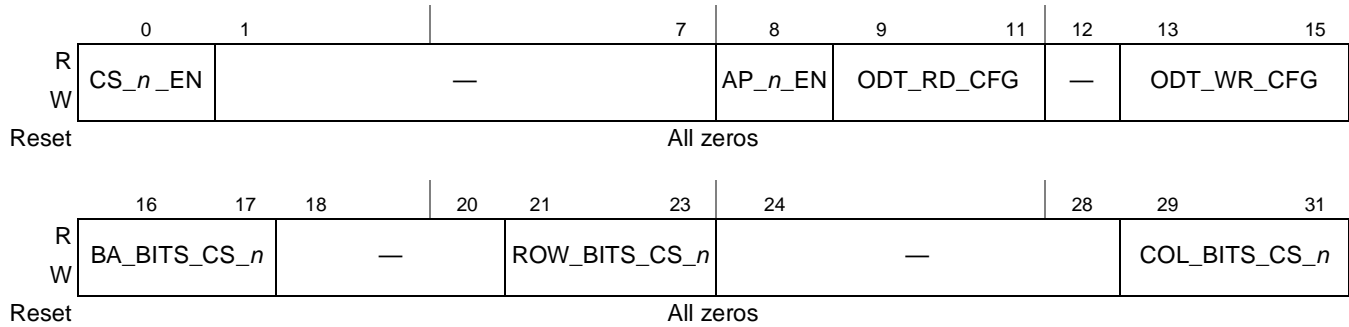
The chip select configuration (CS<sub>*n*</sub>\_CONFIG) registers shown in Figure 9-3 enable the DDR chip selects and set the number of row and column bits used for each chip select. These registers should be loaded with the correct number of row and column bits for each SDRAM. Because CS<sub>*n*</sub>\_CONFIG[ROW\_BITS\_CS\_*n*, 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

Access: Read/Write



**Figure 9-3. Chip Select Configuration Register (CS<sub>n</sub>\_CONFIG)**

Table 9-7 describes the CS<sub>n</sub>\_CONFIG register fields.

**Table 9-7. CS<sub>n</sub>\_CONFIG Field Descriptions**

Bits	Name	Description
0	CS <sub>n</sub> _EN	Chip select <i>n</i> enable 0 Chip select <i>n</i> is not active 1 Chip select <i>n</i> is active and assumes the state set in CS <sub>n</sub> _BNDS.
1–7	—	Reserved
8	AP <sub>n</sub> _EN	Chip select <i>n</i> auto-precharge enable 0 Chip select <i>n</i> is only auto-precharged if global auto-precharge mode is enabled (DDR_SDRAM_INTERVAL[BSTOPRE] = 0). 1 Chip select <i>n</i> 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. 000 Never assert ODT for reads 001 Assert ODT only during reads to CS <sub>n</sub> 010 Assert ODT only during reads to other chip selects 011 Reserved 100 Assert ODT for all reads 101–111Reserved
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. 000 Never assert ODT for writes 001 Assert ODT only during writes to CS <sub>n</sub> 010 Assert ODT only during writes to other chip selects 011 Reserved 100 Assert ODT for all writes 101–111Reserved
16–17	BA_BITS_CS <sub>n</sub>	Number of bank bits for SDRAM on chip select <i>n</i> . These bits correspond to the sub-bank bits driven on MBA <sub>n</sub> in Table 9-38. 00 2 logical bank bits 01 3 logical bank bits 10–11Reserved

Table 9-7. CS<sub>n</sub>\_CONFIG Field Descriptions (continued)

Bits	Name	Description
18–20	—	Reserved
21–23	ROW_BITS_CS <sub>n</sub>	Number of row bits for SDRAM on chip select <i>n</i> . See Table 9-38 for details. 000 12 row bits 001 13 row bits 010 14 row bits 011–111 Reserved
24–28	—	Reserved
29–31	COL_BITS_CS <sub>n</sub>	Number of column bits for SDRAM on chip select <i>n</i> . For DDR, the decoding is as follows: 000 8 column bits 001 9 column bits 010 10 column bits 011 11 column bits 100–111 Reserved

### 9.4.1.3 DDR SDRAM Timing Configuration 3 (TIMING\_CFG\_3)

DDR SDRAM timing configuration register 3, shown in Figure 9-4, sets the extended refresh recovery time, which is combined with TIMING\_CFG\_1[REFREC] to determine the full refresh recovery time.

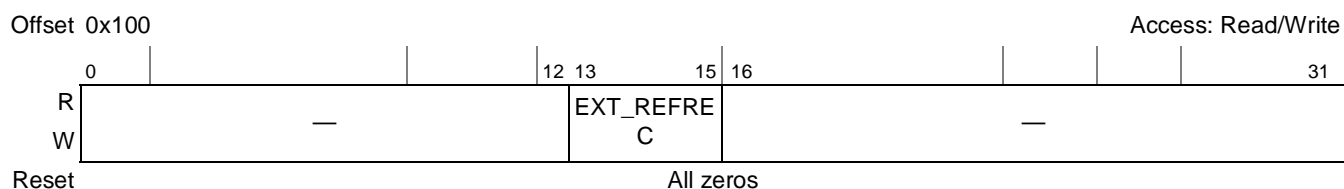


Figure 9-4. DDR SDRAM Timing Configuration 3 (TIMING\_CFG\_3)

Table 9-8 describes TIMING\_CFG\_3 fields.

Table 9-8. TIMING\_CFG\_3 Field Descriptions

Bits	Name	Description
0–12	—	Reserved, should be cleared.
13–15	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 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. $t_{RFC} = \{EXT\_REFREC \parallel REFREC\} + 8$ , such that $t_{RFC}$ is calculated as follows: 000 0 clocks 001 16 clocks 010 32 clocks 011 48 clocks 100 64 clocks 101 80 clocks 110 96 clocks 111 112 clocks
16–31	—	Reserved, should be cleared.

### 9.4.1.4 DDR SDRAM Timing Configuration 0 (TIMING\_CFG\_0)

DDR SDRAM timing configuration register 0, shown in Figure 9-5, sets the number of clock cycles between various SDRAM control commands.

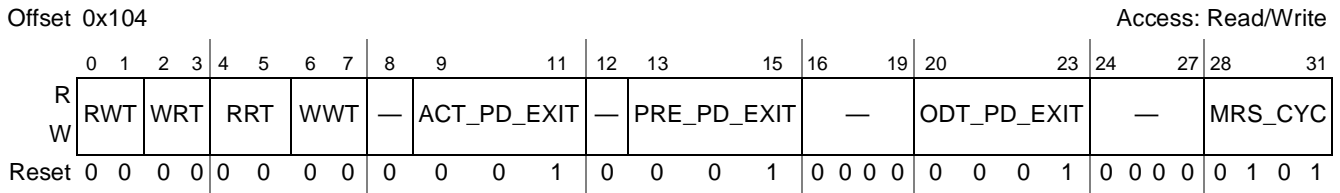


Figure 9-5. DDR SDRAM Timing Configuration 0 (TIMING\_CFG\_0)

Table 9-9 describes TIMING\_CFG\_0 fields.

Table 9-9. TIMING\_CFG\_0 Field Descriptions

Bits	Name	Description
0–1	RWT	Read-to-write turnaround ( $t_{RWT}$ ). 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 $CL - WL + BL \div 2 + 2$ . 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.  00 0 clocks <span style="float: right;">10 2 clocks</span> 01 1 clock <span style="float: right;">11 3 clocks</span>
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 $WL - CL + BL \div 2 + 1$ . 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.  00 0 clocks <span style="float: right;">10 2 clocks</span> 01 1 clock <span style="float: right;">11 3 clocks</span>
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.  00 0 clocks <span style="float: right;">10 2 clocks</span> 01 1 clock <span style="float: right;">11 3 clocks</span>
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.  00 0 clocks <span style="float: right;">10 2 clocks</span> 01 1 clock <span style="float: right;">11 3 clocks</span>
8	—	Reserved, should be cleared.

Table 9-9. TIMING\_CFG\_0 Field Descriptions (continued)

Bits	Name	Description
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. 000 Reserved 001 1 clock 010 2 clocks 011 3 clocks 100 4 clocks 101 5 clocks 110 6 clocks 111 7 clocks
12	—	Reserved, should be cleared.
13–15	PRE_PD_EXIT	Precharge powerdown exit timing ( $t_{XP}$ ). Specifies how many clock cycles to wait after exiting precharge powerdown before issuing any command. 000 Reserved 001 1 clock 010 2 clocks 011 3 clocks 100 4 clocks 101 5 clocks 110 6 clocks 111 7 clocks
16–19	—	Reserved, should be cleared.
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. 0000 0 clock 0001 1 clock 0010 2 clocks 0011 3 clocks 0100 4 clocks 0101 5 clocks 0110 6 clocks 0111 7 clocks 1000 8 clocks 1001 9 clocks 1010 10 clocks 1011 11 clocks 1100 12 clocks 1101 13 clocks 1110 14 clocks 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. 0000 Reserved 0001 1 clock 0010 2 clocks 0011 3 clocks 0100 4 clocks 0101 5 clocks 0110 6 clocks 0111 7 clocks 1000 8 clocks 1001 9 clocks 1010 10 clocks 1011 11 clocks 1100 12 clocks 1101 13 clocks 1110 14 clocks 1111 15 clocks

### 9.4.1.5 DDR SDRAM Timing Configuration 1 (TIMING\_CFG\_1)

DDR SDRAM timing configuration register 1, shown in Figure 9-6, sets the number of clock cycles between various SDRAM control commands.

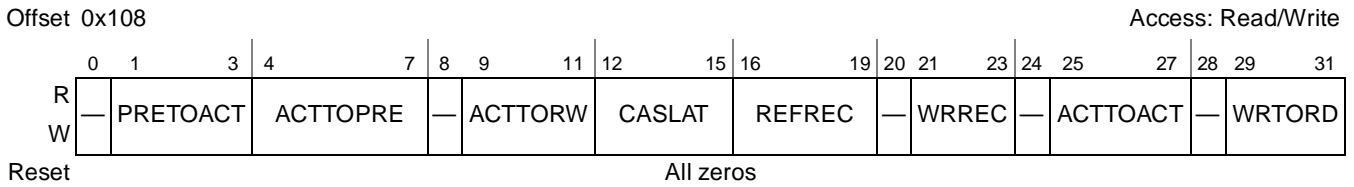


Figure 9-6. DDR SDRAM Timing Configuration 1 (TIMING\_CFG\_1)

Table 9-10 describes TIMING\_CFG\_1 fields.

Table 9-10. TIMING\_CFG\_1 Field Descriptions

Bits	Name	Description
0	—	Reserved, should be cleared.
1–3	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.  000 Reserved 001 1 clock 010 2 clocks 011 3 clocks 100 4 clocks 101 5 clocks 110 6 clocks 111 7 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.  0000 16 clocks                      0101 5 clocks 0001 17 clocks                      0110 6 clocks 0010 18 clocks                      0111 7 clocks 0011 19 clocks                      ... 0100 4 clocks                        1111 15 clocks
8	—	Reserved, should be cleared.
9–11	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.  000 Reserved 001 1 clock 010 2 clocks 011 3 clocks 100 4 clocks 101 5 clocks 110 6 clocks 111 7 clocks

Table 9-10. TIMING\_CFG\_1 Field Descriptions (continued)

Bits	Name	Description																																
12–15	CASLAT	<p><math>\overline{MCAS}</math> 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 <math>n</math> and the latency is <math>m</math> clocks, data is available nominally coincident with clock edge <math>n + m</math>. This value must be programmed at initialization as described in <a href="#">Section 9.4.1.8, “DDR SDRAM Control Configuration 2 (DDR_SDRAM_CFG_2).”</a></p> <table> <tr> <td>0000</td> <td>Reserved</td> <td>1000</td> <td>4.5 clocks</td> </tr> <tr> <td>0001</td> <td>Reserved</td> <td>1001</td> <td>5 clocks</td> </tr> <tr> <td>0010</td> <td>Reserved</td> <td>1010</td> <td>5.5 clocks</td> </tr> <tr> <td>0011</td> <td>Reserved</td> <td>1011</td> <td>6 clocks</td> </tr> <tr> <td>0100</td> <td>Reserved</td> <td>1100</td> <td>6.5 clocks</td> </tr> <tr> <td>0101</td> <td>3 clocks</td> <td>1101</td> <td>7 clocks</td> </tr> <tr> <td>0110</td> <td>3.5 clocks</td> <td>1110</td> <td>7.5 clocks</td> </tr> <tr> <td>0111</td> <td>4 clocks</td> <td>1111</td> <td>8 clocks</td> </tr> </table>	0000	Reserved	1000	4.5 clocks	0001	Reserved	1001	5 clocks	0010	Reserved	1010	5.5 clocks	0011	Reserved	1011	6 clocks	0100	Reserved	1100	6.5 clocks	0101	3 clocks	1101	7 clocks	0110	3.5 clocks	1110	7.5 clocks	0111	4 clocks	1111	8 clocks
0000	Reserved	1000	4.5 clocks																															
0001	Reserved	1001	5 clocks																															
0010	Reserved	1010	5.5 clocks																															
0011	Reserved	1011	6 clocks																															
0100	Reserved	1100	6.5 clocks																															
0101	3 clocks	1101	7 clocks																															
0110	3.5 clocks	1110	7.5 clocks																															
0111	4 clocks	1111	8 clocks																															
16–19	REFREC	<p>Refresh recovery time (<math>t_{RFC}</math>). 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 <math>t_{RFC}</math> is calculated as follows: <math>t_{RFC} = \{EXT\_REFREC \parallel REFREC\} + 8</math>.</p> <table> <tr> <td>0000</td> <td>8 clocks</td> <td>0011</td> <td>11 clocks</td> </tr> <tr> <td>0001</td> <td>9 clocks</td> <td>...</td> <td></td> </tr> <tr> <td>0010</td> <td>10 clocks</td> <td>1111</td> <td>23 clocks</td> </tr> </table>	0000	8 clocks	0011	11 clocks	0001	9 clocks	...		0010	10 clocks	1111	23 clocks																				
0000	8 clocks	0011	11 clocks																															
0001	9 clocks	...																																
0010	10 clocks	1111	23 clocks																															
20	—	Reserved, should be cleared.																																
21–23	WRREC	<p>Last data to precharge minimum interval (<math>t_{WR}</math>). Determines the number of clock cycles from the last data associated with a write command until a precharge command is allowed.</p> <table> <tr> <td>000</td> <td>Reserved</td> </tr> <tr> <td>001</td> <td>1 clock</td> </tr> <tr> <td>010</td> <td>2 clocks</td> </tr> <tr> <td>011</td> <td>3 clocks</td> </tr> <tr> <td>100</td> <td>4 clocks</td> </tr> <tr> <td>101</td> <td>5 clocks</td> </tr> <tr> <td>110</td> <td>6 clocks</td> </tr> <tr> <td>111</td> <td>7 clocks</td> </tr> </table>	000	Reserved	001	1 clock	010	2 clocks	011	3 clocks	100	4 clocks	101	5 clocks	110	6 clocks	111	7 clocks																
000	Reserved																																	
001	1 clock																																	
010	2 clocks																																	
011	3 clocks																																	
100	4 clocks																																	
101	5 clocks																																	
110	6 clocks																																	
111	7 clocks																																	
24	—	Reserved, should be cleared.																																
25–27	ACTTOACT	<p>Activate-to-activate interval (<math>t_{RRD}</math>). 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).</p> <table> <tr> <td>000</td> <td>Reserved</td> <td>100</td> <td>4 clocks</td> </tr> <tr> <td>001</td> <td>1 clock</td> <td>101</td> <td>5 clocks</td> </tr> <tr> <td>010</td> <td>2 clocks</td> <td>110</td> <td>6 clocks</td> </tr> <tr> <td>011</td> <td>3 clocks</td> <td>111</td> <td>7 clocks</td> </tr> </table>	000	Reserved	100	4 clocks	001	1 clock	101	5 clocks	010	2 clocks	110	6 clocks	011	3 clocks	111	7 clocks																
000	Reserved	100	4 clocks																															
001	1 clock	101	5 clocks																															
010	2 clocks	110	6 clocks																															
011	3 clocks	111	7 clocks																															
28	—	Reserved, should be cleared.																																
29–31	WRTORD	<p>Last write data pair to read command issue interval (<math>t_{WTR}</math>). Number of clock cycles between the last write data pair and the subsequent read command to the same physical bank.</p> <table> <tr> <td>000</td> <td>Reserved</td> <td>100</td> <td>4 clocks</td> </tr> <tr> <td>001</td> <td>1 clock</td> <td>101</td> <td>5 clocks</td> </tr> <tr> <td>010</td> <td>2 clocks</td> <td>110</td> <td>6 clocks</td> </tr> <tr> <td>011</td> <td>3 clocks</td> <td>111</td> <td>7 clocks</td> </tr> </table>	000	Reserved	100	4 clocks	001	1 clock	101	5 clocks	010	2 clocks	110	6 clocks	011	3 clocks	111	7 clocks																
000	Reserved	100	4 clocks																															
001	1 clock	101	5 clocks																															
010	2 clocks	110	6 clocks																															
011	3 clocks	111	7 clocks																															

### 9.4.1.6 DDR SDRAM Timing Configuration 2 (TIMING\_CFG\_2)

DDR SDRAM timing configuration 2, shown in Figure 9-7, sets the clock delay to data for writes.

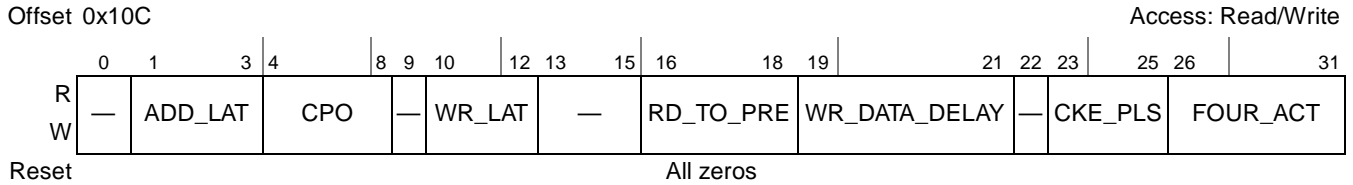


Figure 9-7. DDR SDRAM Timing Configuration 2 Register (TIMING\_CFG\_2)

Table 9-11 describes the TIMING\_CFG\_2 fields.

Table 9-11. TIMING\_CFG\_2 Field Descriptions

Bits	Name	Description
0	—	Reserved
1–3	ADD_LAT	Additive latency. The additive latency must be set to a value less than TIMING_CFG_1[ACTTORW]. 000 0 clocks 001 1 clock 010 2 clocks 011 3 clocks 100 4 clocks 101 5 clocks 110 Reserved 111 Reserved
4–8	CPO	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. 00000READ_LAT + 1                      01100READ_LAT + 5/2 00001Reserved                            01101READ_LAT + 11/4 00010READ_LAT                            01110READ_LAT + 3 00011READ_LAT + 1/4                    01111READ_LAT + 13/4 00100READ_LAT + 1/2                    10000READ_LAT + 7/2 00101READ_LAT + 3/4                    10001READ_LAT + 15/4 00110READ_LAT + 1                      10010READ_LAT + 4 00111READ_LAT + 5/4                    10011READ_LAT + 17/4 01000READ_LAT + 3/2                    10100READ_LAT + 9/2 01001READ_LAT + 7/4                    10101READ_LAT + 19/4 01010READ_LAT + 2                      10110–11111 Reserved 01011READ_LAT + 9/4
9	—	Reserved
10–12	WR_LAT	Write latency. Note that the total write latency for DDR2 is equal to WR_LAT + ADD_LAT. 000 Reserved 001 1 clock 010 2 clocks 011 3 clocks 100 4 clocks 101 5 clocks 110 6 clocks 111 7 clocks
13–15	—	Reserved



Table 9-11. TIMING\_CFG\_2 Field Descriptions (continued)

Bits	Name	Description
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. 000 Reserved 100 4 cycles 001 1 cycle 101–111 Reserved 010 2 cycles 011 3 cycles
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 9.5.7, “DDR SDRAM Write Timing Adjustments,” for details. 000 0 clock delay 100 1 clock delay 001 1/4 clock delay 101 5/4 clock delay 010 1/2 clock delay 110 3/2 clock delay 011 3/4 clock delay 111 Reserved
22	—	Reserved
23–25	CKE_PLS	Minimum CKE pulse width ( $t_{CKE}$ ). 000 Reserved 011 3 cycles 001 1 cycle 100 4 cycles 010 2 cycles 101–111 Reserved
26–31	FOUR_ACT	Window for four activates ( $t_{FAW}$ ). This is applied to DDR2 with eight logical banks only. 000000 Reserved ... 0000011 cycle 010011 19 cycles 0000102 cycles 01010020 cycles 0000113 cycles 010101–111111 Reserved 0001004 cycles

### 9.4.1.7 DDR SDRAM Control Configuration (DDR\_SDRAM\_CFG)

The DDR SDRAM control configuration register, shown in Figure 9-8, enables the interface logic and specifies certain operating features such as self refreshing, error checking and correcting, registered DIMMs, and dynamic power management.

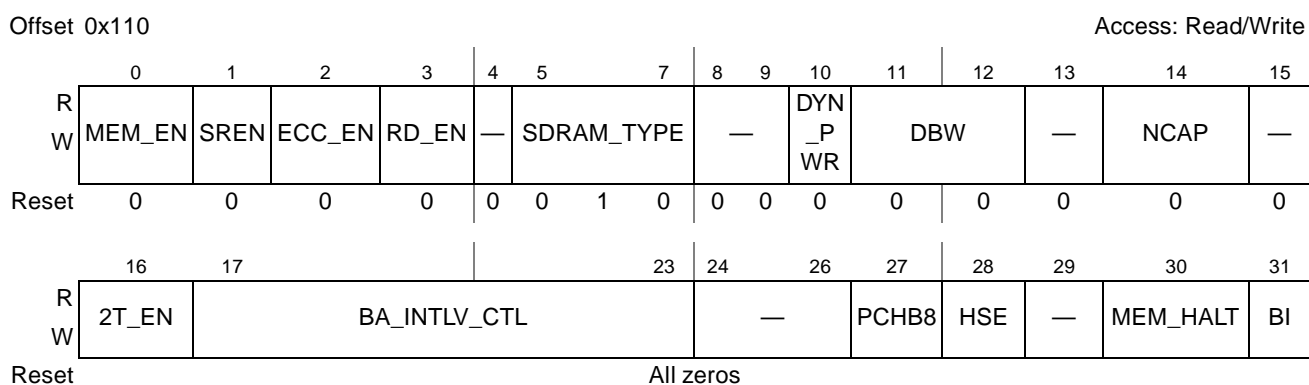


Figure 9-8. DDR SDRAM Control Configuration Register (DDR\_SDRAM\_CFG)

Table 9-12 describes the DDR\_SDRAM\_CFG fields.

**Table 9-12. DDR\_SDRAM\_CFG Field Descriptions**

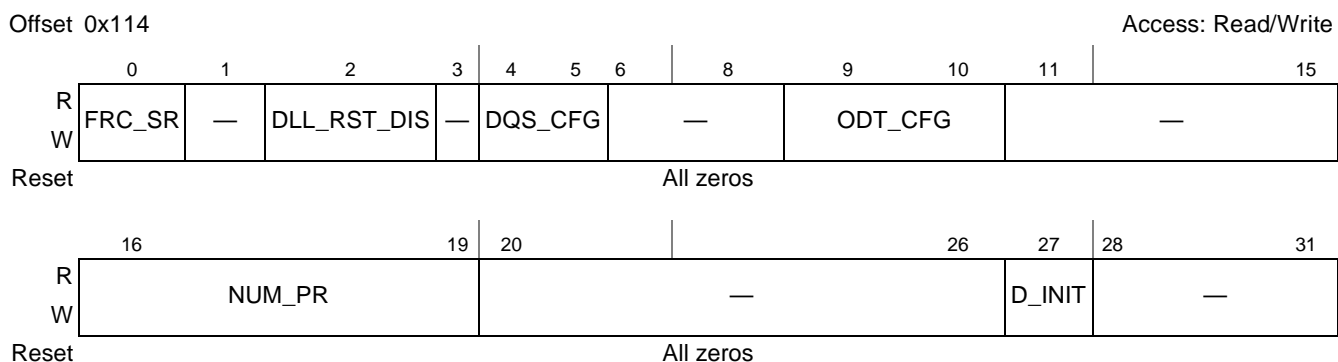
Bits	Name	Description
0	MEM_EN	DDR SDRAM interface logic enable. 0 SDRAM interface logic is disabled. 1 SDRAM interface logic is enabled. Must not be set until all other memory configuration parameters have been appropriately configured by initialization code.
1	SREN	Self refresh enable (during sleep). 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. 1 SDRAM self refresh is enabled during sleep.
2	ECC_EN	ECC enable. Note that uncorrectable read errors may cause an interrupt. 0 No ECC errors are reported. No ECC interrupts are generated. 1 ECC is enabled.
3	RD_EN	Registered DRAM module enable. Specifies the type of DRAM module used in the system. 0 Indicates unbuffered DRAM modules. 1 Indicates registered DRAM modules. <b>Note:</b> 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. For DDR2 SDRAM, the field is set to 011.
8–9	—	Reserved
10	DYN_PWR	Dynamic power management mode 0 Dynamic power management mode is disabled. 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. 00 Reserved 01 32-bit bus is used 10 16-bit bus is used 11 Reserved
13	—	Reserved
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. 0 DRAMs in system support concurrent auto-precharge. 1 DRAMs in system do not support concurrent auto-precharge.
15	—	Reserved
16	2T_EN	Enable 2T timing. 0 1T timing is enabled. The DRAM command/address are held for only 1 cycle on the DRAM bus. 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. <b>Note:</b> RD_EN and 2T_EN must not both be set at the same time.

Table 9-12. DDR\_SDRAM\_CFG Field Descriptions (continued)

Bits	Name	Description
17–23	BA_INTLV_CTL	Bank (chip select) interleaving control. Set this field only if you wish to use bank interleaving. (All unlisted field values are reserved.) 0000000No external memory banks are interleaved 1000000External memory banks 0 and 1 are interleaved
24–26	—	Reserved
27	PCHB8	Precharge bit 8 enable. 0 MA[10] is used to indicate the auto-precharge and precharge all commands. 1 MA[8] is used to indicate the auto-precharge and precharge all commands.
28	HSE	Global half-strength override Sets I/O driver impedance to half strength. This impedance is used by the 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 <a href="#">Section 5.2.2.9, "DDR Control Driver Register (DDRCDR)."</a> This bit should be cleared if using automatic hardware calibration. 0 I/O driver impedance is configured to full strength. 1 I/O driver impedance is configured to half strength.
29	—	Reserved
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. 0 DDR controller accepts new transactions. 1 DDR controller finishes any remaining transactions, and then it remains halted until this bit is cleared by software.
31	BI	Bypass initialization 0 DDR controller cycles through initialization routine based on SDRAM_TYPE 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. See <a href="#">Section 9.4.1.15, "DDR Initialization Address (DDR_INIT_ADDR),"</a> for details on avoiding ECC errors in this mode.

### 9.4.1.8 DDR SDRAM Control Configuration 2 (DDR\_SDRAM\_CFG\_2)

The DDR SDRAM control configuration register 2, shown in [Figure 9-9](#), provides more control configuration for the DDR controller.



**Figure 9-9. DDR SDRAM Control Configuration Register 2 (DDR\_SDRAM\_CFG\_2)**

[Table 9-13](#) describes the DDR\_SDRAM\_CFG\_2 fields.

**Table 9-13. DDR\_SDRAM\_CFG\_2 Field Descriptions**

Bits	Name	Description
0	FRC_SR	Force self refresh 0 DDR controller operates in normal mode. 1 DDR controller enters self-refresh mode.
1	—	Reserved. Should be cleared.
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. 0 DDR controller issues a DLL reset to the DRAMs when exiting self refresh. 1 DDR controller does not issue a DLL reset to the DRAMs when exiting self refresh.
3	—	Reserved
4–5	DQS_CFG	DQS configuration 00 Only true DQS signals are used. 01 Reserved 10 Reserved 11 Reserved
6–8	—	Reserved
9–10	ODT_CFG	ODT configuration. This field defines how ODT is driven to the on-chip IOs. See <a href="#">Section 5.2.2.9, “DDR Control Driver Register (DDRCDR),”</a> which defines the termination value that is used. 00 Never assert ODT to internal IOs 01 Assert ODT to internal IOs only during writes to DRAM 10 Assert ODT to internal IOs only during reads to DRAM 11 Always keep ODT asserted to internal IOs
11–15	—	Reserved.

Table 9-13. DDR\_SDRAM\_CFG\_2 Field Descriptions (continued)

Bits	Name	Description
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. 0000 Reserved 0001 1 refresh is issued at a time 0010 2 refreshes is issued at a time 0011 3 refreshes is issued at a time ... 1000 8 refreshes is issued at a time 1001–1111 Reserved
20–26	—	Reserved, should be cleared.
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. 0 There is not data initialization in progress, and no data initialization is scheduled 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.
28–31	—	Reserved

#### 9.4.1.9 DDR SDRAM Mode Configuration (DDR\_SDRAM\_MODE)

The DDR SDRAM mode configuration register, shown in Figure 9-10, sets the values loaded into the DDR's mode registers.

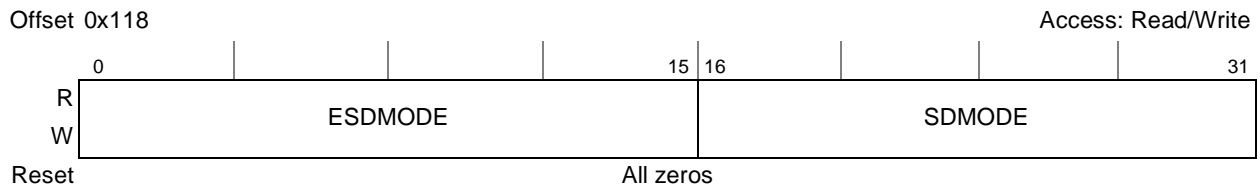


Figure 9-10. DDR SDRAM Mode Configuration Register (DDR\_SDRAM\_MODE)

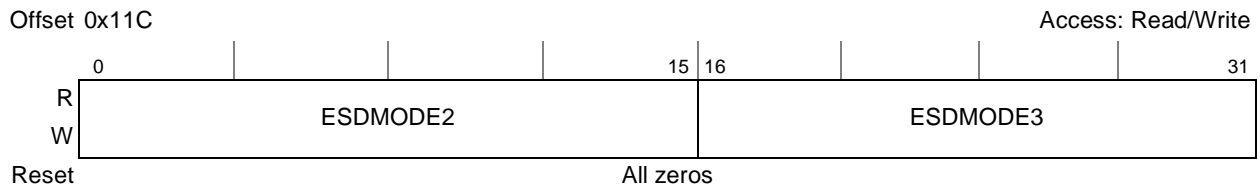
Table 9-14 describes the DDR\_SDRAM\_MODE fields.

**Table 9-14. 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. 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 9-10, corresponds to ESDMODE[15]. The msb of the SDRAM extended mode register value must be stored at ESDMODE[0].
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. 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 9-10, 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].

### 9.4.1.10 DDR SDRAM Mode 2 Configuration (DDR\_SDRAM\_MODE\_2)

The DDR SDRAM mode 2 configuration register, shown in Figure 9-11, sets the values loaded into the DDR's extended mode 2 and 3 registers (for DDR2).



**Figure 9-11. DDR SDRAM Mode 2 Configuration Register (DDR\_SDRAM\_MODE\_2)**

Table 9-15 describes the DDR\_SDRAM\_MODE\_2 fields.

**Table 9-15. DDR\_SDRAM\_MODE\_2 Field Descriptions**

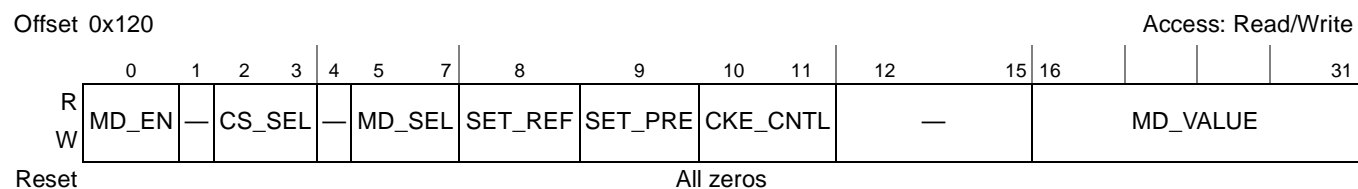
Bits	Name	Description
0–15	ESDMODE2	Extended SDRAM mode 2. Specifies the initial value loaded into the DDR SDRAM extended 2 mode register. The range and meaning of legal values is specified by the DDR SDRAM manufacturer. When this value is driven onto the address bus (during the DDR SDRAM initialization sequence), MA[0] presents the lsb bit of ESDMODE2, which, in the big-endian convention shown in Figure 9-11, 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 9-11, corresponds to ESDMODE3[15]. The msb of the SDRAM extended mode 3 register value must be stored at ESDMODE3[0].

### 9.4.1.11 DDR SDRAM Mode Control Register (DDR\_SDRAM\_MD\_CNTL)

The DDR SDRAM mode control register, shown in [Figure 9-12](#), 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 9-16](#) describes the fields of this register. [Table 9-17](#) shows the user how to set the fields of this register to accomplish the above tasks.



**Figure 9-12. DDR SDRAM Mode Control Register (DDR\_SDRAM\_MD\_CNTL)**

[Table 9-16](#) 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 9-16. 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: <ul style="list-style-type: none"> <li>• MODE REGISTER SET</li> <li>• EXTENDED MODE REGISTER SET</li> <li>• EXTENDED MODE REGISTER SET 2</li> <li>• EXTENDED MODE REGISTER SET 3</li> </ul> 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. 0 Indicates that no mode register set command needs to be issued. 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. 00 Chip select 0 is active 01 Chip select 1 is active 10 Reserved 11 Reserved
4	—	Reserved

**Table 9-16. DDR\_SDRAM\_MD\_CNTL Field Descriptions (continued)**

Bits	Name	Description
5–7	MD_SEL	<p>Mode register select. MD_SEL specifies one of the following:</p> <ul style="list-style-type: none"> <li>• During a mode select command, selects the SDRAM mode register to be changed</li> <li>• During a precharge command, selects the SDRAM logical bank to be precharged. A precharge all command ignores this field.</li> <li>• During a refresh command, this field is ignored.</li> </ul> <p>Note that MD_SEL contains the value that is presented onto the memory bank address pins (MBA<sub>n</sub>) of the DDR controller.</p> <p>000 MR 001 EMR 010 EMR2 011 EMR3</p>
8	SET_REF	<p>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.</p> <p>0 Indicates that no refresh command needs to be issued. 1 Indicates that a refresh command is ready to be issued.</p>
9	SET_PRE	<p>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.</p> <p>0 Indicates that no precharge all command needs to be issued. 1 Indicates that a precharge all command is ready to be issued.</p>
10–11	CKE_CNTL	<p>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).</p> <p>00 CKE signals are not forced by software. 01 CKE signals are forced to a low value by software. 10 CKE signals are forced to a high value by software. 11 Reserved</p>
12–15	—	Reserved
16–31	MD_VALUE	<p>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.</p> <p>For a mode register set command, this field contains the data to be written to the selected mode register. For a precharge command, only bit five is significant:</p> <p>0 Issue a precharge command; MD_SEL selects the logical bank to be precharged 1 Issue a precharge all command; all logical banks are precharged</p>

Table 9-17 shows how DDR\_SDRAM\_MD\_CNTL fields should be set for each of the tasks described above.

**Table 9-17. 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	—



Table 9-17. Settings of DDR\_SDRAM\_MD\_CNTL Fields (continued)

Field	Mode Register Set	Refresh	Precharge	Clock Enable Signals Control
CS_SEL	Chooses chip select (CS)			—
MD_SEL	Select mode register. See Table 9-16.	—	Selects logical bank	—
MD_VALUE	Value written to mode register	—	Only bit five is significant. See Table 9-16.	—
CKE_CNTL	0	0	0	See Table 9-16.

#### 9.4.1.12 DDR SDRAM Interval Configuration (DDR\_SDRAM\_INTERVAL)

The DDR SDRAM interval configuration register, shown in Figure 9-13, 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.

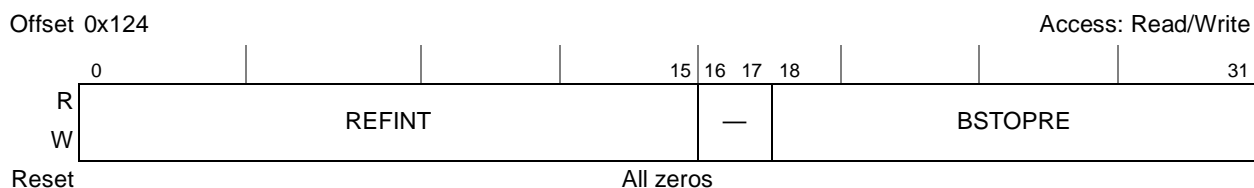


Figure 9-13. DDR SDRAM Interval Configuration Register (DDR\_SDRAM\_INTERVAL)

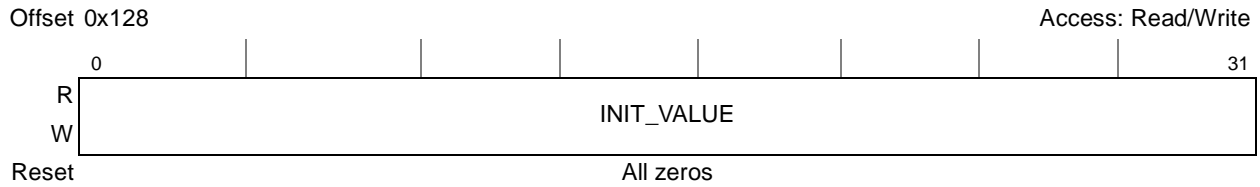
Table 9-18 describes the DDR\_SDRAM\_INTERVAL fields.

Table 9-18. 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.

### 9.4.1.13 DDR SDRAM Data Initialization (DDR\_DATA\_INIT)

The DDR SDRAM data initialization register, shown in [Figure 9-14](#), provides the value that is used to initialize memory if DDR\_SDRAM\_CFG2[D\_INIT] is set.



**Figure 9-14. DDR SDRAM Data Initialization Configuration Register (DDR\_DATA\_INIT)**

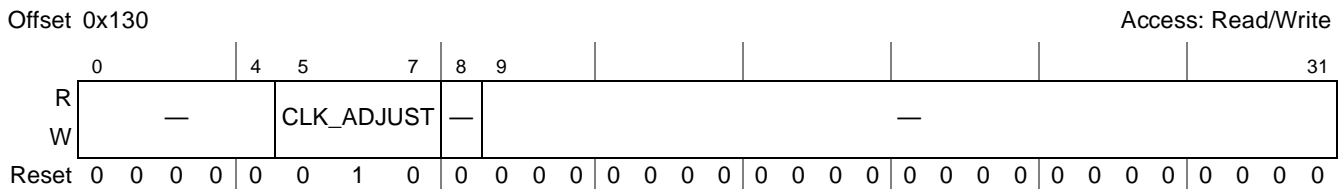
[Table 9-19](#) describes the DDR\_DATA\_INIT fields.

**Table 9-19. 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.

### 9.4.1.14 DDR SDRAM Clock Control (DDR\_SDRAM\_CLK\_CNTL)

The DDR SDRAM clock control configuration register, shown in [Figure 9-15](#), provides a 1/4-cycle clock adjustment.



**Figure 9-15. DDR SDRAM Clock Control Configuration Register (DDR\_SDRAM\_CLK\_CNTL)**

[Table 9-20](#) describes the DDR\_SDRAM\_CLK\_CNTL fields.

**Table 9-20. DDR\_SDRAM\_CLK\_CNTL Field Descriptions**

Bits	Name	Description
0–4	—	Reserved
5–7	CLK_ADJUST	Clock adjust. 000 Clock is launched aligned with address/command 001 Clock is launched 1/4 applied cycle after address/command 010 Clock is launched 1/2 applied cycle after address/command 011 Clock is launched 3/4 applied cycle after address/command 100 Clock is launched 1 applied cycle after address/command 101–111 Reserved
8	—	Reserved, should be cleared.
9–31	—	Reserved

### 9.4.1.15 DDR Initialization Address (DDR\_INIT\_ADDR)

The DDR SDRAM initialization address register, shown in Figure 9-16, provides the address that is used for the automatic CAS to preamble calibration after POR.

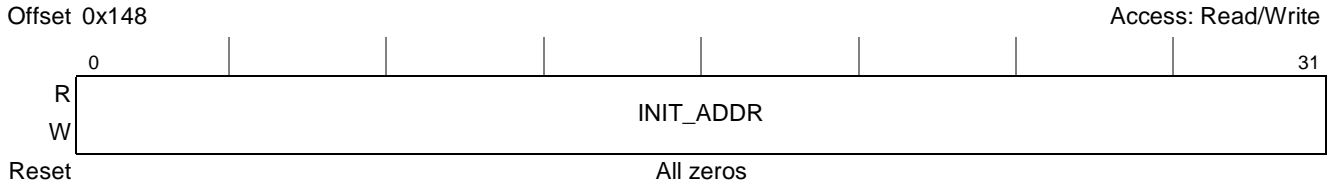


Figure 9-16. DDR Initialization Address Configuration Register (DDR\_INIT\_ADDR)

Table 9-21 describes the DDR\_INIT\_ADDR fields.

Table 9-21. DDR\_INIT\_ADDR Field Descriptions

Bits	Name	Description
0–31	INIT_ADDR	Initialization address. Represents the address that is used for the automatic CAS to preamble calibration at POR.

### 9.4.1.16 DDR IP Block Revision 1 (DDR\_IP\_REV1)

The DDR IP block revision 1 register, shown in Figure 9-17, provides read-only fields with the IP block ID, along with major and minor revision information.

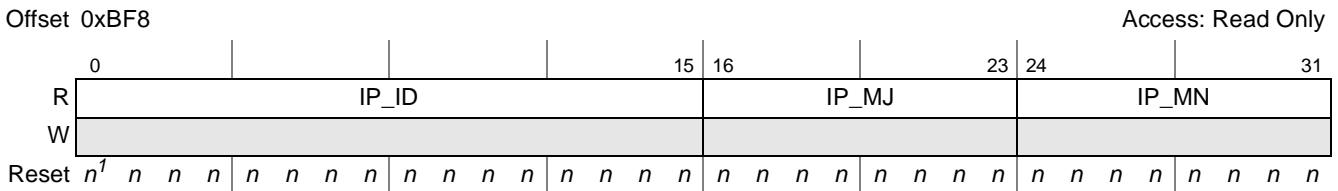


Figure 9-17. DDR IP Block Revision 1 (DDR\_IP\_REV1)

<sup>1</sup> For reset values, see Table 9-22.

Table 9-22 describes the DDR\_IP\_REV1 fields.

Table 9-22. 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 0x02.
24–31	IP_MN	Minor revision. This is currently set to 0x01.

### 9.4.1.17 DDR IP Block Revision 2 (DDR\_IP\_REV2)

The DDR IP block revision 2 register, shown in Figure 9-18, provides read-only fields with the IP block integration and configuration options.

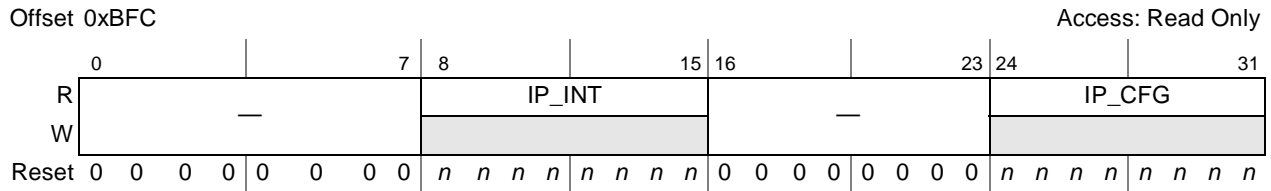


Figure 9-18. DDR IP Block Revision 2 (DDR\_IP\_REV2)

Table 9-23 describes the DDR\_IP\_REV2 fields.

Table 9-23. DDR\_IP\_REV2 Field Descriptions

Bits	Name	Description
0–7	—	Reserved
8–15	IP_INT	IP block integration options. This is currently set to 0x0000.
16–23	—	Reserved
24–31	IP_CFG	IP block configuration options. This is currently set to 0x82.

### 9.4.1.18 Memory Data Path Error Injection Mask High (DATA\_ERR\_INJECT\_HI)

The memory data path error injection mask high register is shown in Figure 9-19.

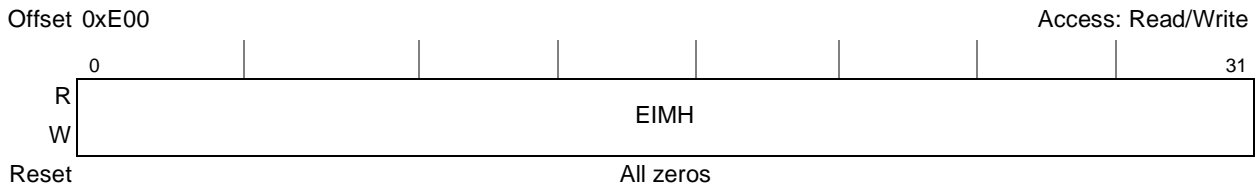


Figure 9-19. Memory Data Path Error Injection Mask High Register (DATA\_ERR\_INJECT\_HI)

Table 9-24 describes the DATA\_ERR\_INJECT\_HI fields.

Table 9-24. 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.

### 9.4.1.19 Memory Data Path Error Injection Mask Low (DATA\_ERR\_INJECT\_LO)

The memory data path error injection mask low register is shown in Figure 9-20.

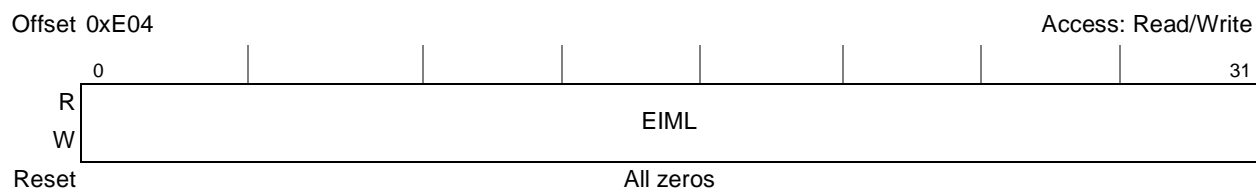


Figure 9-20. Memory Data Path Error Injection Mask Low Register (DATA\_ERR\_INJECT\_LO)

Table 9-25 describes the DATA\_ERR\_INJECT\_LO fields.

Table 9-25. 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.

### 9.4.1.20 Memory Data Path Error Injection Mask ECC (ERR\_INJECT)

The memory data path error injection mask ECC register, shown in Figure 9-21, sets the ECC mask, enables errors to be written to ECC memory, and allows the ECC byte to mirror the most significant data byte.



Figure 9-21. Memory Data Path Error Injection Mask ECC Register (ERR\_INJECT)

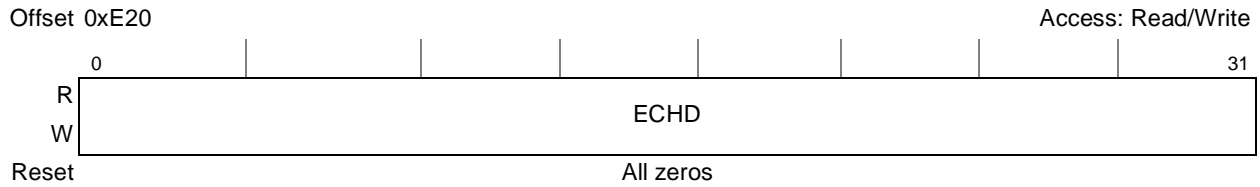
Table 9-26 describes the ERR\_INJECT fields.

Table 9-26. ERR\_INJECT Field Descriptions

Bits	Name	Description
0–21	—	Reserved
22	EMB	ECC mirror byte 0 Mirror byte functionality disabled. 1 Mirror the most significant data path byte onto the ECC byte.
23	EIEN	Error injection enable 0 Error injection disabled. 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.

### 9.4.1.21 Memory Data Path Read Capture High (CAPTURE\_DATA\_HI)

The memory data path read capture high register, shown in [Figure 9-22](#), stores the high word of the read data path during error capture.



**Figure 9-22. Memory Data Path Read Capture High Register (CAPTURE\_DATA\_HI)**

[Table 9-27](#) describes the CAPTURE\_DATA\_HI fields.

**Table 9-27. 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.

### 9.4.1.22 Memory Data Path Read Capture Low (CAPTURE\_DATA\_LO)

The memory data path read capture low register, shown in [Figure 9-23](#), stores the low word of the read data path during error capture.



**Figure 9-23. Memory Data Path Read Capture Low Register (CAPTURE\_DATA\_LO)**

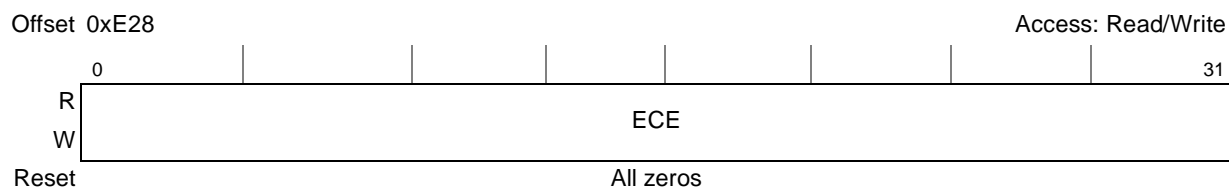
[Table 9-28](#) describes the CAPTURE\_DATA\_LO fields.

**Table 9-28. 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.

### 9.4.1.23 Memory Data Path Read Capture ECC (CAPTURE\_ECC)

The memory data path read capture ECC register, shown in [Figure 9-24](#), stores the ECC syndrome bits that were on the data bus when an error was detected.



**Figure 9-24. Memory Data Path Read Capture ECC Register (CAPTURE\_ECC)**

[Table 9-29](#) describes the CAPTURE\_ECC fields.

**Table 9-29. CAPTURE\_ECC Field Descriptions**

Bits	Name	Description
0–31	ECC	Reserved <ul style="list-style-type: none"> <li>0–7: 8-bit ECC for first 16 bits in 16-bit bus mode; should be ignored for 32-bit</li> <li>8–15: 8-bit ECC for second 16 bits in 16-bit bus mode; 1st 32 bits in 32-bit bus mode</li> <li>16–23: 8-bit ECC for third 16 bits in 16-bit bus mode; should be ignored for 32-bit</li> <li>24–31: 8-bit ECC for fourth 16 bits in 16-bit bus mode; 2nd 32 bits in 32-bit bus mode</li> </ul>

### 9.4.1.24 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 9-25](#).



**Figure 9-25. Memory Error Detect Register (ERR\_DETECT)**

[Table 9-30](#) describes the ERR\_DETECT fields.

**Table 9-30. ERR\_DETECT Field Descriptions**

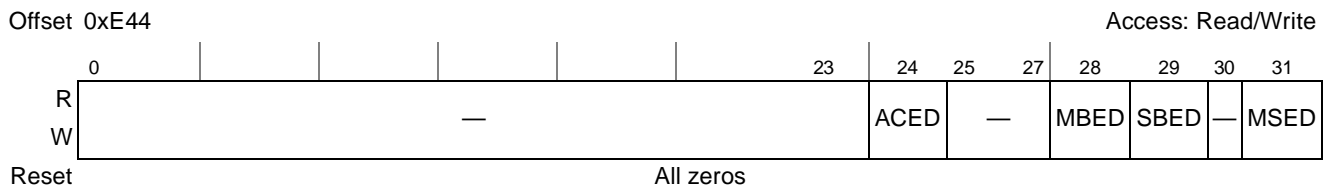
Bits	Name	Description
0	MME	Multiple memory errors. This bit is cleared by software writing a 1. 0 Multiple memory errors of the same type were not detected. 1 Multiple memory errors of the same type were detected.
1–23	—	Reserved

**Table 9-30. ERR\_DETECT Field Descriptions (continued)**

Bits	Name	Description
24	ACE	Automatic calibration error. This bit is cleared by software writing a 1. 0 An automatic calibration error has not been detected. 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. 0 A multiple-bit error has not been detected. 1 A multiple-bit error has been detected.
29	SBE	Single-bit ECC error. This bit is cleared by software writing a 1. 0 The number of single-bit ECC errors detected has not crossed the threshold set in ERR_SBE[SBET]. 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. 0 A memory select error has not been detected. 1 A memory select error has been detected.

### 9.4.1.25 Memory Error Disable (ERR\_DISABLE)

The memory error disable register, shown in [Figure 9-26](#), allows selective disabling of the DDR controller’s error detection circuitry. Disabled errors are not detected or reported.



**Figure 9-26. Memory Error Disable Register (ERR\_DISABLE)**

[Table 9-31](#) describes the ERR\_DISABLE fields.

**Table 9-31. ERR\_DISABLE Field Descriptions**

Bits	Name	Description
0–23	—	Reserved
24	ACED	Automatic calibration error disable 0 Automatic calibration errors are enabled. 1 Automatic calibration errors are disabled.
25–27	—	Reserved
28	MBED	Multiple-bit ECC error disable 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. 1 Multiple-bit ECC errors are not detected or reported.
29	SBED	Single-bit ECC error disable 0 Single-bit ECC errors are enabled. 1 Single-bit ECC errors are disabled.

Downloaded from [Elcodis.com](http://Elcodis.com) electronic components distributor



Table 9-31. ERR\_DISABLE Field Descriptions (continued)

Bits	Name	Description
30	—	Reserved
31	MSED	Memory select error disable 0 Memory select errors are enabled. 1 Memory select errors are disabled.

#### 9.4.1.26 Memory Error Interrupt Enable (ERR\_INT\_EN)

The memory error interrupt enable register, shown in Figure 9-27, enables ECC interrupts or memory select error interrupts. When an enabled interrupt condition occurs, the internal *int* signal is asserted to the programmable interrupt controller (PIC).

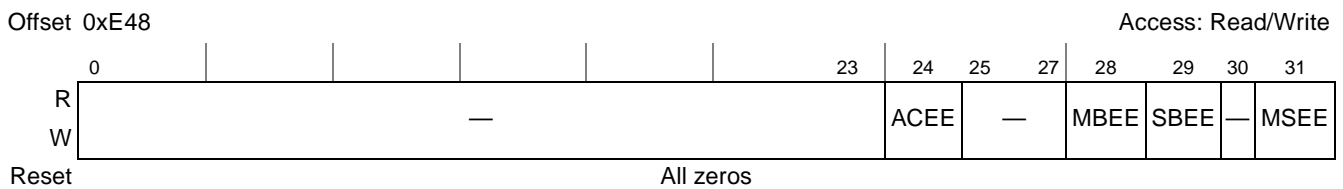


Figure 9-27. Memory Error Interrupt Enable Register (ERR\_INT\_EN)

Table 9-32 describes the ERR\_INT\_EN fields.

Table 9-32. ERR\_INT\_EN Field Descriptions

Bits	Name	Description
0–23	—	Reserved
24	ACEE	Automatic calibration error interrupt enable 0 Automatic calibration errors cannot generate interrupts. 1 Automatic calibration errors generate interrupts.
25–27	—	Reserved
28	MBEE	Multiple-bit ECC error interrupt enable. 0 Multiple-bit ECC errors cannot generate interrupts. 1 Multiple-bit ECC errors generate interrupts.
29	SBEE	Single-bit ECC error interrupt enable 0 Single-bit ECC errors cannot generate interrupts. 1 Single-bit ECC errors generate interrupts.
30	—	Reserved
31	MSEE	Memory select error interrupt enable 0 Memory select errors do not cause interrupts. 1 Memory select errors generate interrupts.

### 9.4.1.27 Memory Error Attributes Capture (CAPTURE\_ATTRIBUTES)

The memory error attributes capture register, shown in Figure 9-28, sets attributes for errors including type, size, source, and others.

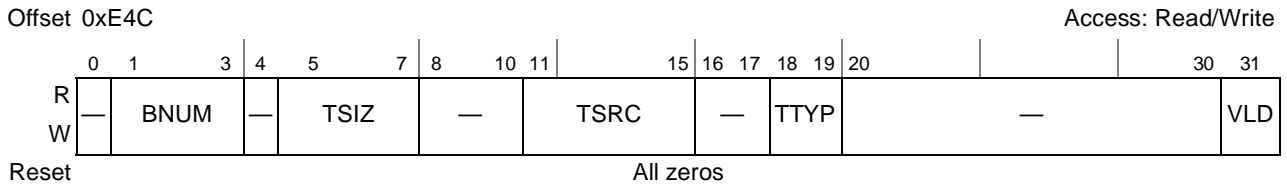


Figure 9-28. Memory Error Attributes Capture Register (CAPTURE\_ATTRIBUTES)

Table 9-33 describes the CAPTURE\_ATTRIBUTES fields.

Table 9-33. 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. 000 4 double words 001 1 double word 010 2 double words 011 3 double words Others Reserved
8–10	—	Reserved
11–15	TSRC	Transaction source for the error  00000 e300 core data transaction                    01001 I <sup>2</sup> C (boot sequencer) 00001 Reserved    01010 JTAG 00010 e300 core instruction fetch                    01011 Reserved 00011 Reserved    01100 eSDHC 00100 eTSEC1    01101–11100 Reserved 00101 eTSEC2    11101 PCI Express 00110 Reserved    11110 Reserved 00111 USB    11111 DMA 01000 Reserved
16–17	—	Reserved
18–19	TTYP	Transaction type for the error. 00 Reserved 01 Write 10 Read 11 Read-modify-write
20–30	—	Reserved
31	VLD	Valid. Set as soon as valid information is captured in the error capture registers.

### 9.4.1.28 Memory Error Address Capture (CAPTURE\_ADDRESS)

The memory error address capture register, shown in Figure 9-29, holds the 32 lsbs of a transaction when a DDR ECC error is detected.

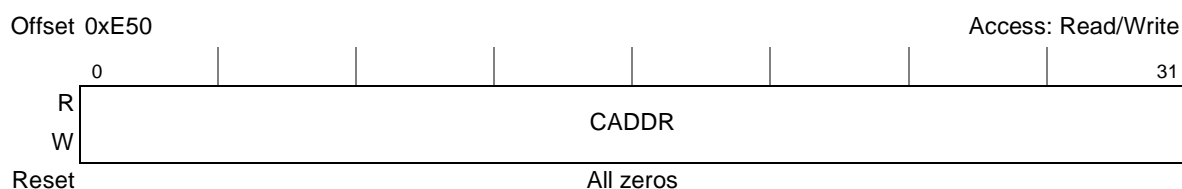


Figure 9-29. Memory Error Address Capture Register (CAPTURE\_ADDRESS)

Table 9-34 describes the CAPTURE\_ADDRESS fields.

Table 9-34. 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.

### 9.4.1.29 Single-Bit ECC Memory Error Management (ERR\_SBE)

The single-bit ECC memory error management register, shown in Figure 9-30, 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 9-30. Single-Bit ECC Memory Error Management Register (ERR\_SBE)

Table 9-35 describes the ERR\_SBE fields.

Table 9-35. 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. As a special case, setting this to zero means error threshold is set to 256.
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 an interrupt is generated when this value equals SBET. SBEC is automatically cleared when the threshold value is reached.

## 9.5 Functional Description

The DDR SDRAM controller controls processor and I/O interactions with system memory. It provides support for JEDEC-compliant DDR2 SDRAM. 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.

Figure 9-1 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 two physical banks of 16-/24-/32-/40-bit wide memory. Bank sizes up to 512 Mbytes are supported, providing up to a maximum of 1 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 32-bit data bus, detects all double-bit errors within the 32-bit data bus, and detects all errors within a nibble. The controller allows as many as 16 pages to be open simultaneously. The amount of time (in clock cycles) the pages remain open is programmable with `DDR_SDRAM_INTERVAL[BSTOPRE]`.

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) 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 specifies 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[0:3]`) 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 9-31 shows an example DDR SDRAM configuration with four logical banks.

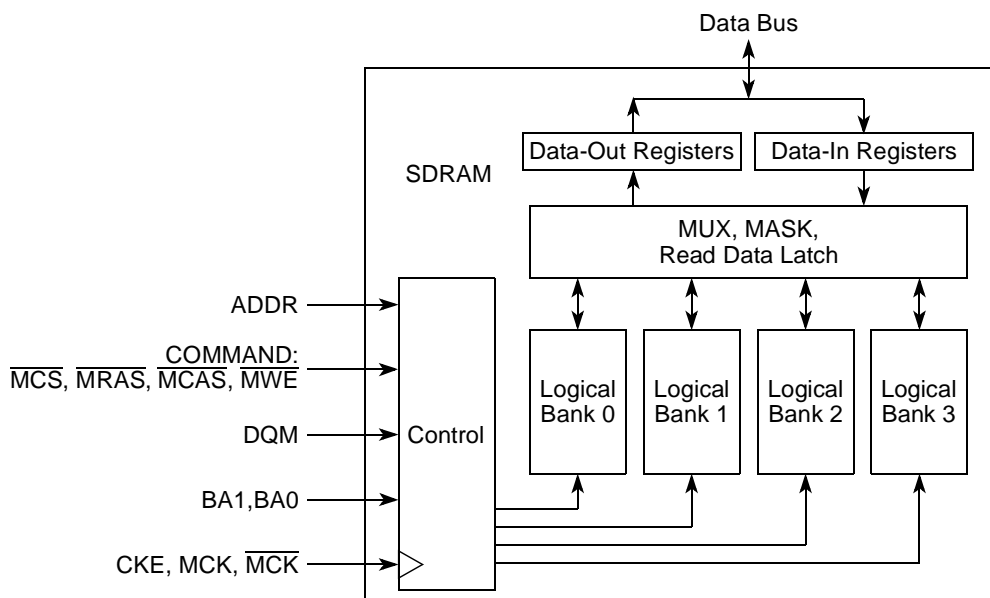


Figure 9-31. Typical Dual Data Rate SDRAM Internal Organization

Figure 9-32 shows some typical signal connections.

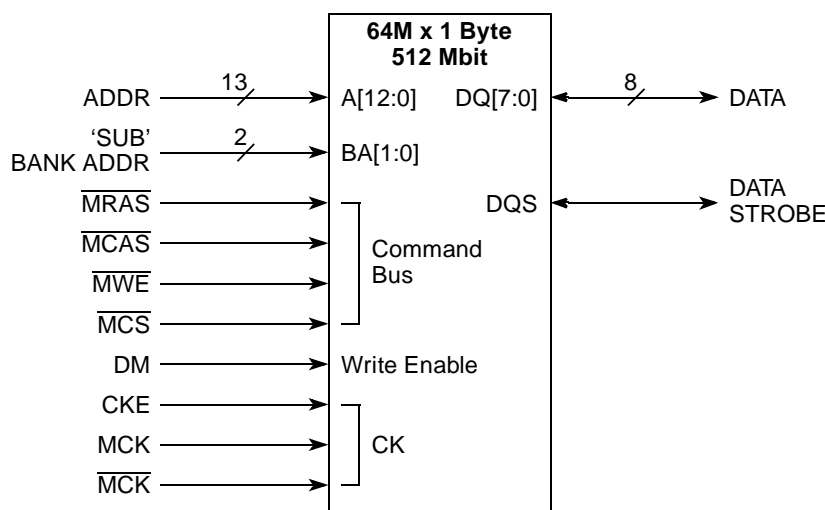
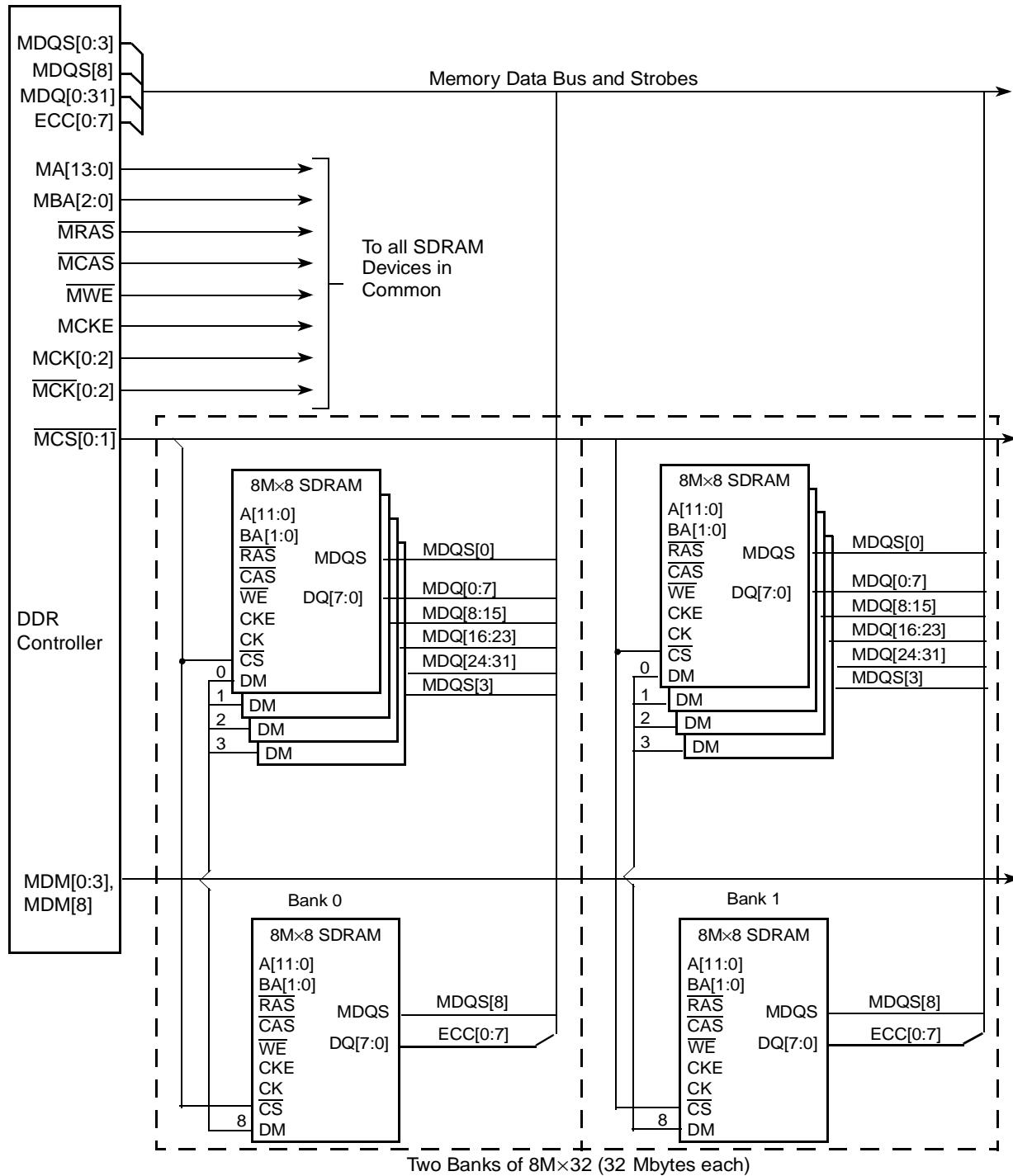


Figure 9-32. Typical DDR SDRAM Interface Signals

Figure 9-33 shows an example DDR SDRAM configuration with two physical banks each comprised of four  $8\text{M} \times 8$  DDR modules for a total of 64 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 14 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[0:1], MCK[0:2], MCK[0:2], MDM[0:3], MDM[8], and the data bus signals.
2. Each of the MCS[0:1] signals correspond with a separate physical bank of memory.
3. Buffering may be needed if large memory arrays are used.

**Figure 9-33. Example 64-Mbyte DDR SDRAM Configuration With ECC**

For information on how the DDR2 memory controller handles errors, see [Section 9.5.12, “Error Management.”](#)

## 9.5.1 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. Fourteen multiplexed address signals and three logical bank select signals support device densities from 64 Mbits to 2 Gbits. Two chip select (CS) signals support two banks of DIMM 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 32 bits wide, 40 bits with ECC. The DDR memory controller supports physical bank sizes from 16 Mbytes to 512 Mbytes. The physical banks can be constructed using  $\times 8$ ,  $\times 16$ , or  $\times 32$  memory devices. The memory technologies supported are 64 Mbits, 128 Mbits, 256 Mbits, 512 Mbits, and 1 Gbit. Some 2-Gbit devices are supported depending on the internal device configuration. 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 9-36](#) shows the DDR memory controller’s relationships between data byte lane0–3, MDM[0:3], MDQS[0:3], and MDQ[0:31] when DDR SDRAM memories are used with  $\times 8$  or  $\times 16$  devices.

**Table 9-36. Byte Lane to Data Relationship**

Data Byte Lane	Data Bus Mask	Data Bus Strobe	Data Bus
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]

### 9.5.1.1 Supported DDR SDRAM Organizations

Although the DDR memory controller multiplexes row and column address bits onto 14 memory address signals and 3 logical bank select signals, a physical bank may be implemented with memory devices requiring fewer than 30 address bits. The physical bank may be configured to provide from 12 to 14 row address bits, plus 2 to 3 logical bank-select bits and from 8–11 column address bits.

[Table 9-37](#) describe DDR SDRAM device configurations supported by the DDR memory controller.

### NOTE

DDR SDRAM is limited to 30 total address bits.

**Table 9-37. Supported DDR2 SDRAM Device Configurations**

SDRAM Device	Device Configuration	Row × Column × Bank Bits	32-Bit Bank Size	Two Banks of Memory
256 Mbits	32 Mbits × 8	13 × 10 × 2	128 Mbytes	256 Mbytes
256 Mbits	16 Mbits × 16	13 × 9 × 2	64 Mbytes	128 Mbytes
512 Mbits	64 Mbits × 8	14 × 10 × 2	256 Mbytes	512 Mbytes
512 Mbits	32 Mbits × 16	13 × 10 × 2	128 Mbytes	256 Mbytes
1 Gbits	128 Mbits × 8	14 × 10 × 3	512 Mbytes	1 Gbyte
1 Gbits	64 Mbits × 16	13 × 10 × 3	256 Mbytes	512 Mbytes
2 Gbits	128 Mbits × 16	14 × 10 × 3	512 Mbytes	1 Gbytes

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 9.5.12, “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  $MCS_n$  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.

### 9.5.2 DDR SDRAM Address Multiplexing

The following tables ([Table 9-38](#) and [Table 9-39](#)) show the address bit encodings for each DDR SDRAM configuration. The address presented at the memory controller signals MA[13:0] use MA[13] as the msb and MA[0] as the lsb. Also, MA[10] is used as the auto-precharge bit in DDR2 modes for reads and writes, so the column address can never use MA[10].

**Table 9-38. DDR2 Address Multiplexing for 32-Bit Data Bus with Interleaving and Partial Array Self Refresh Disabled**

Row × Col	msb	Address from Core Master																												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	
14 × 10 × 3	MRAS				13	12	11	10	9	8	7	6	5	4	3	2	1	0																
	MBA																		2	1	0													
	MCAS																					9	8	7	6	5	4	3	2	1	0			
14 × 10 × 2	MRAS				13	12	11	10	9	8	7	6	5	4	3	2	1	0																
	MBA																			1	0													
	MCAS																					9	8	7	6	5	4	3	2	1	0			
13 × 10 × 3	MRAS				12	11	10	9	8	7	6	5	4	3	2	1	0																	
	MBA																		2	1	0													
	MCAS																					9	8	7	6	5	4	3	2	1	0			



**Table 9-38. DDR2 Address Multiplexing for 32-Bit Data Bus with Interleaving and Partial Array Self Refresh Disabled (continued)**

Row × Col	msb	Address from Core Master																												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	
13 × 10 × 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 × 9 × 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 9-39. DDR2 Address Multiplexing for 16-Bit Data Bus**

Row × Col	msb	Address from Core Master																														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	
14 × 10 × 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 × 10 × 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 × 10 × 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 × 10 × 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 × 9 × 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. When interleaving is enabled, the chip selects being interleaved must use the same size of memory. One extra bit in the address decode is used for the interleaving to determine which chip select to access.

Table 9-40 illustrates examples of address decode when interleaving between two chip selects.

**Table 9-40. Example of Address Multiplexing for 32-Bit Data Bus Interleaving between Two Banks with Partial Array Self Refresh Disabled**

Row × Col	msb	Address from Core Master																													lsb		
	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			
14 × 10 × 3	MRAS		13	12	11	10	9	8	7	6	5	4	3	2	1	0	CS SEL																
	MBA																	2	1	0													
	MCAS																				9	8	7	6	5	4	3	2	1	0			
14 × 10 × 2	MRAS		13	12	11	10	9	8	7	6	5	4	3	2	1	0	CS SEL																
	MBA																	1	0														
	MCAS																			9	8	7	6	5	4	3	2	1	0				
13 × 10 × 3	MRAS		12	11	10	9	8	7	6	5	4	3	2	1	0	CS SEL																	
	MBA																	2	1	0													
	MCAS																			9	8	7	6	5	4	3	2	1	0				
13 × 10 × 2	MRAS		12	11	10	9	8	7	6	5	4	3	2	1	0	CS SEL																	
	MBA																	1	0														
	MCAS																			9	8	7	6	5	4	3	2	1	0				

### 9.5.3 JEDEC Standard DDR SDRAM Interface Commands

The following section describes the commands and timings the controller uses when operating in DDR2 mode.

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 9-41) 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  $\overline{MCAS}$  before  $\overline{MRAS}$ )

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:  $\overline{MCAS}$  latency, additive latency, write recovery, burst type, and burst length.  $\overline{MCAS}$  latency may be chosen as provided by the preferred SDRAM (some SDRAMs provide  $\overline{MCAS}$  latency {1,2,3}, some provide  $\overline{MCAS}$  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. 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,  $\overline{MCAS}$  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 9-41. DDR SDRAM Command Table**

Operation	CKE Prev.	CKE Current	$\overline{MCS}$	$\overline{MRAS}$	$\overline{MCAS}$	$\overline{MWE}$	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

Table 9-41. DDR SDRAM Command Table (continued)

Operation	CKE Prev.	CKE Current	$\overline{\text{MCS}}$	$\overline{\text{MRAS}}$	$\overline{\text{MCAS}}$	$\overline{\text{MWE}}$	MBA	MA10	MA
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

### 9.5.4 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-beat burst read, but ignores the last three beats. Single-beat writes are performed by masking the last three beats of the four-beat burst using the data mask MDM[0:3]. 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 9-42 with granularity of one memory clock cycle, except for CASLAT, which can be programmed with 1/2 clock granularity.

Table 9-42. 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 $t_{\text{RRD}}$ .
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 $t_{\text{RAS}}$ .
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 $t_{\text{RCD}}$ .
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 $n + m$ .

Table 9-42. DDR SDRAM Interface Timing Intervals (continued)

Timing Intervals	Definition
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 9.4.1.4, “DDR SDRAM Timing Configuration 0 (TIMING\_CFG\_0),” Section 9.4.1.5, “DDR SDRAM Timing Configuration 1 (TIMING\_CFG\_1),” Section 9.4.1.6, “DDR SDRAM Timing Configuration 2 (TIMING\_CFG\_2),” and Section 9.4.1.3, “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 9-34 through Figure 9-36 show DDR SDRAM timing for various types of accesses; see Figure 9-34 for a single-beat read operation, Figure 9-35 for a single-beat write operation, and Figure 9-36 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.

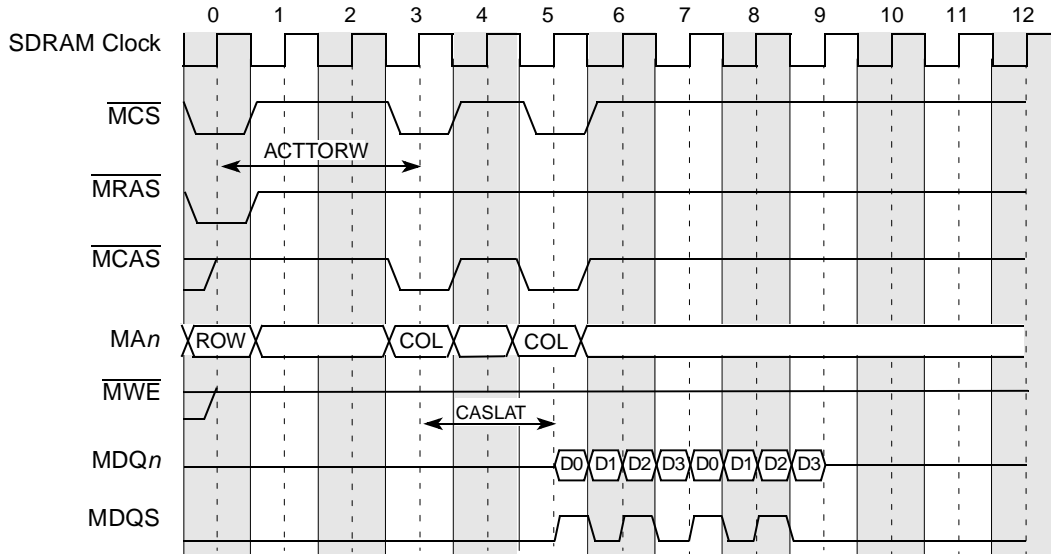


Figure 9-34. DDR SDRAM Burst Read Timing—ACTTORW = 3, MCAS Latency = 2

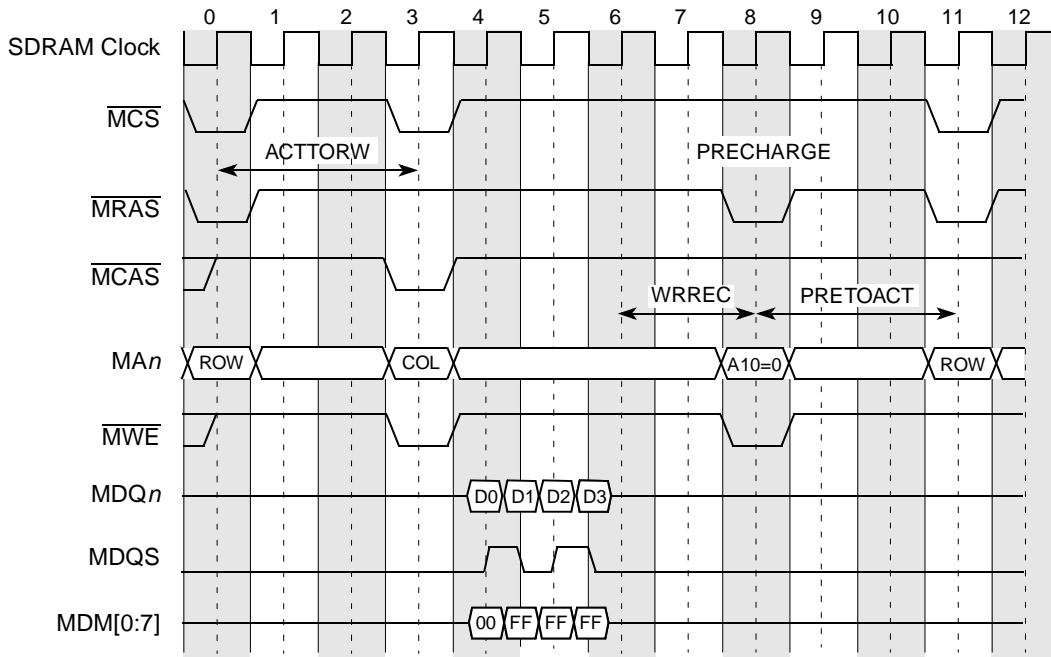


Figure 9-35. DDR SDRAM Single-Beat (Double Word) Write Timing—ACTTOR

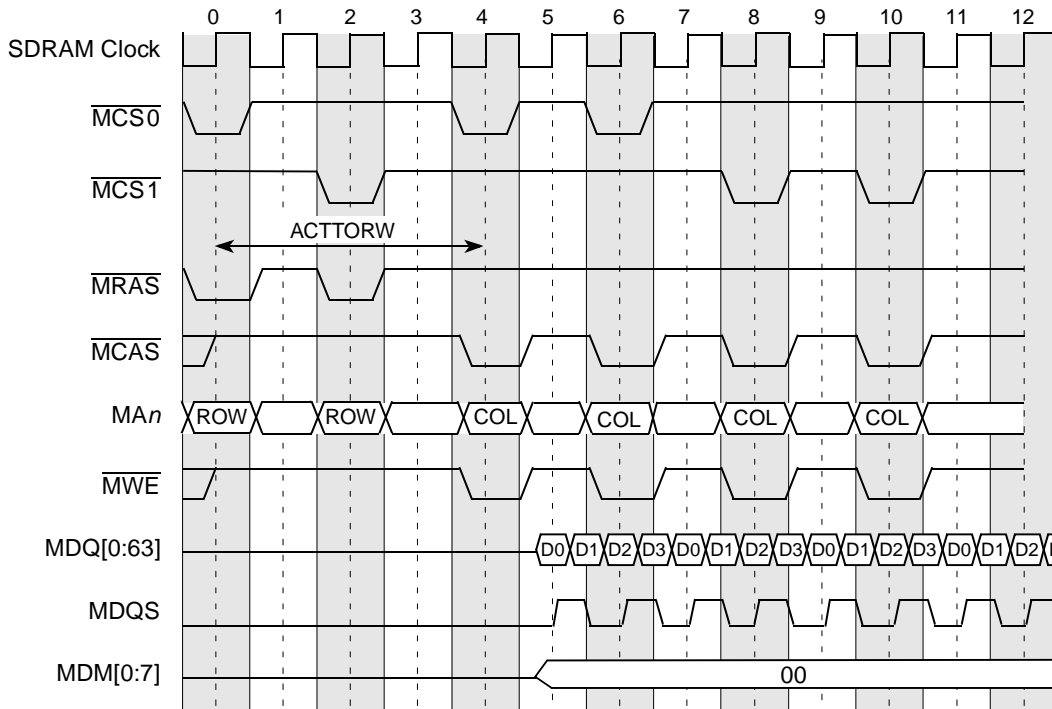


Figure 9-36. DDR SDRAM Single-Beat (Double Word) Write Timing—ACTTORW = 3

### 9.5.4.1 Clock Distribution

The following list discusses recommendations for 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.
- 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.

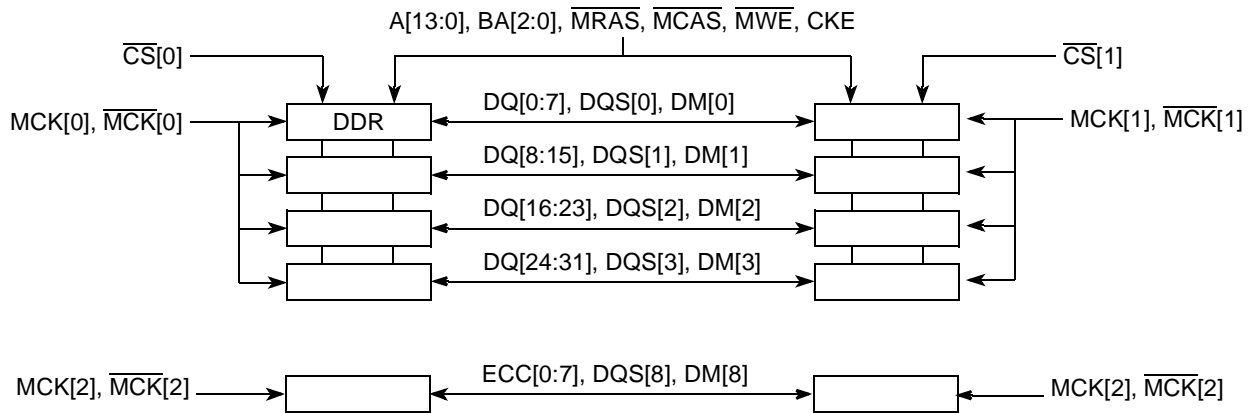


Figure 9-37. DDR SDRAM Clock Distribution Example for 8 DDR SDRAMs

### 9.5.5 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 9-38 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.

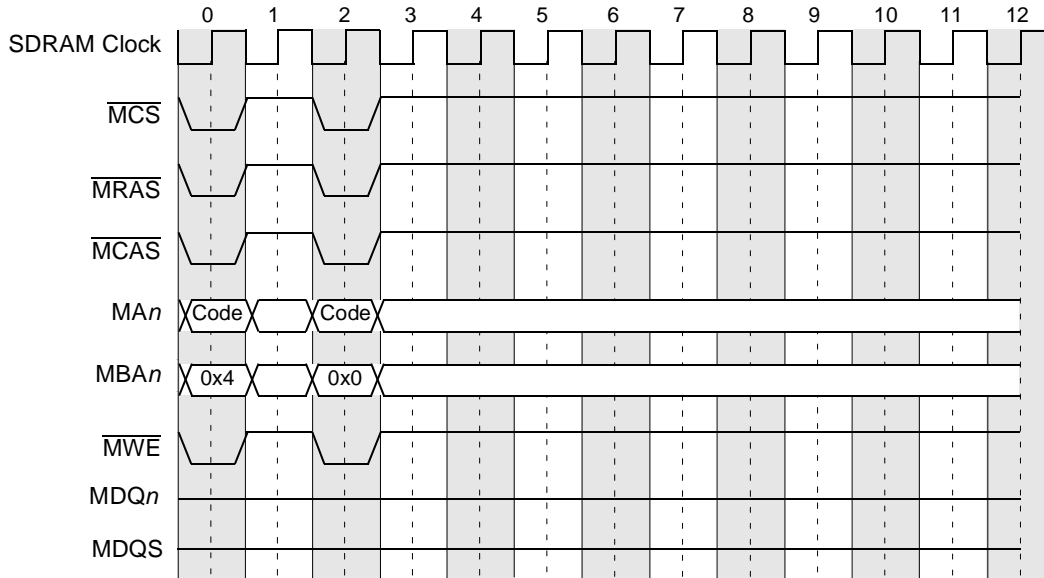


Figure 9-38. DDR SDRAM Mode-Set Command Timing

### 9.5.6 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.



Figure 9-39 shows the registered DDR SDRAM DIMM single-beat write timing.

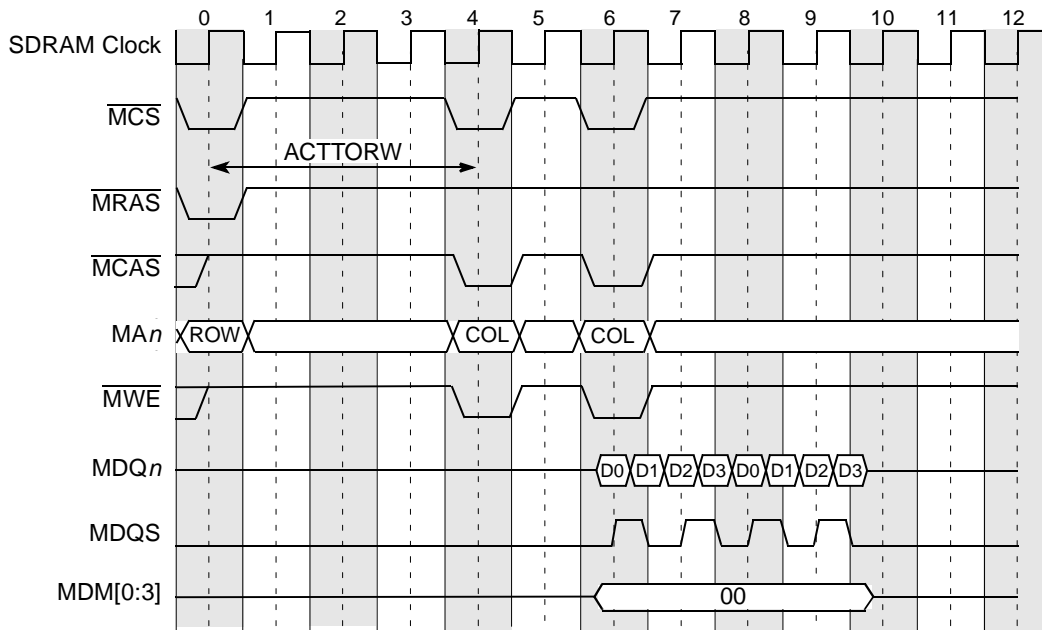


Figure 9-39. Registered DDR SDRAM DIMM Burst Write Timing

### 9.5.7 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. 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 9-40 shows the use of the WR\_DATA\_DELAY parameter.

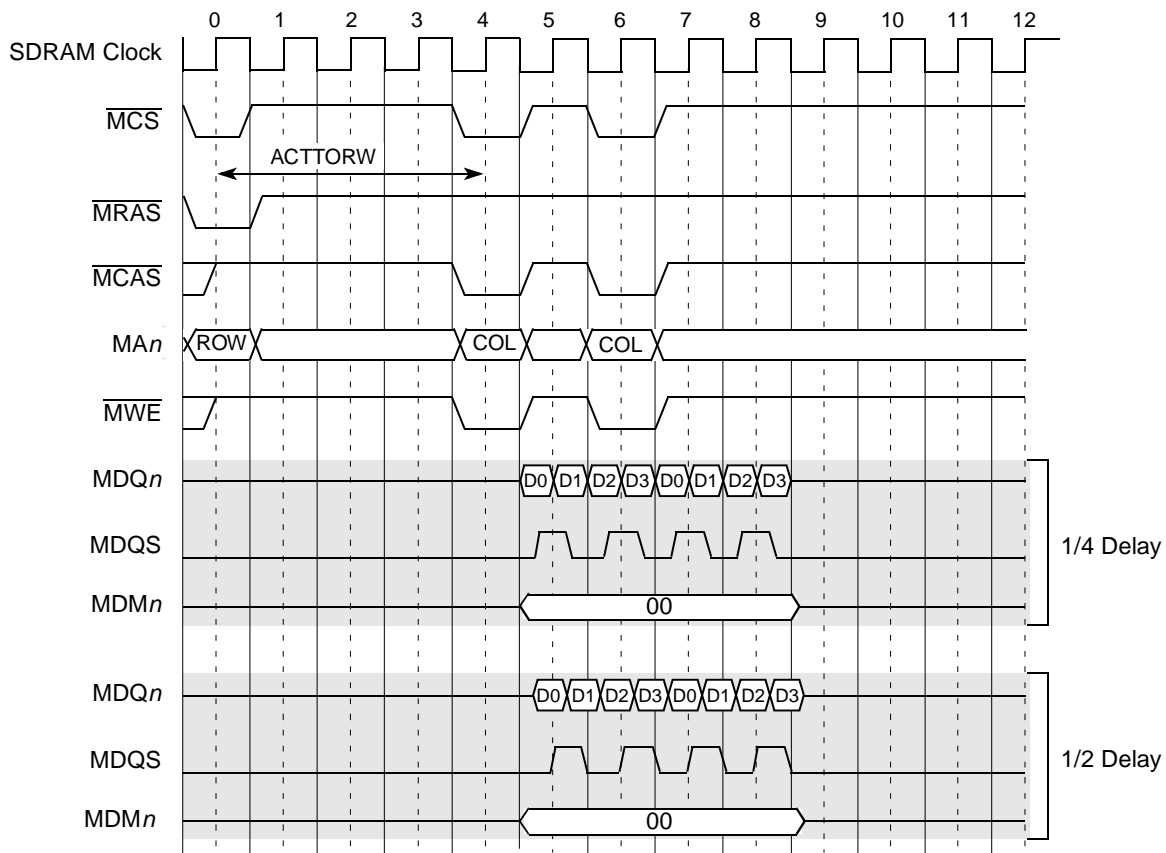


Figure 9-40. Write Timing Adjustments Example for Write Latency = 1

### 9.5.8 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 two 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 populated with one DIMM. 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 two 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.

### 9.5.8.1 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 9-41](#) (`TIMING_CFG_1 [REFREC] = 10` in this example).

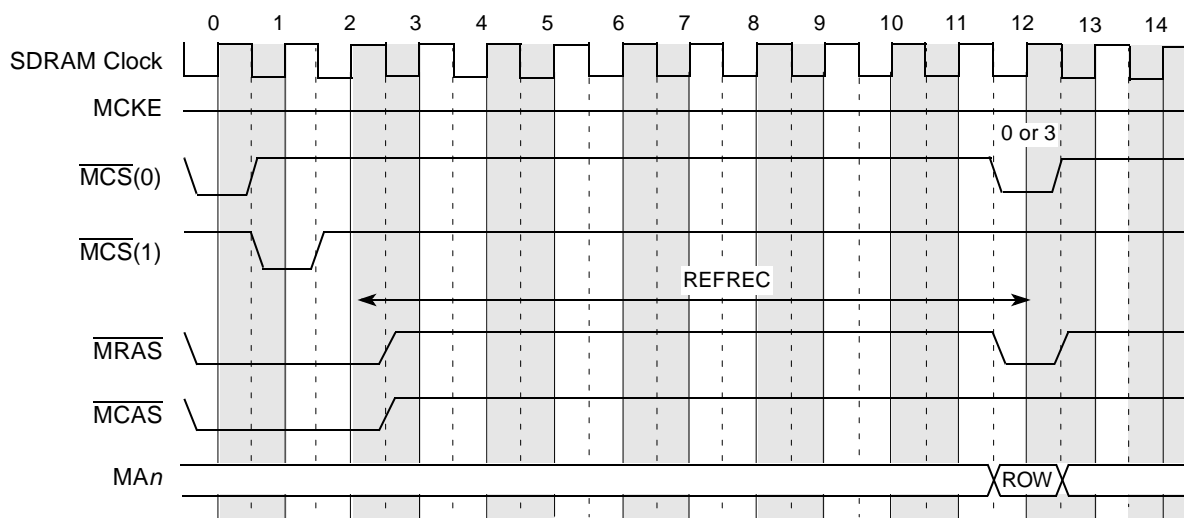


Figure 9-41. DDR SDRAM Bank Staggered Auto Refresh Timing

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.

### 9.5.8.2 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 9-43 summarizes the refresh types available in each power-saving mode.

**Table 9-43. DDR SDRAM Power-Saving Modes Refresh Configuration**

Power Saving Mode	Refresh Type	SREN
Sleep	Self	1
	None	—

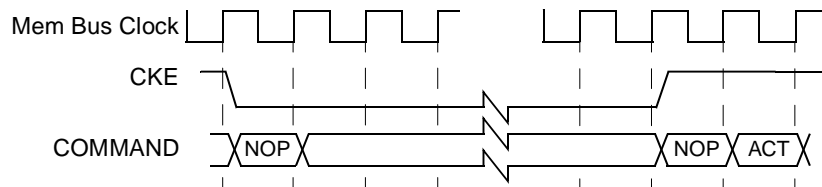
Note that in the absence of refresh support, system software must preserve DDR SDRAM data (such as by copying the data to disk) before entering the power-saving mode.

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 9-42.



**Figure 9-42. DDR SDRAM Power-Down Mode**

### 9.5.8.2.1 Self-Refresh in Sleep Mode

The entry and exit timing for self-refreshing SDRAMs is shown in Figure 9-43 and Figure 9-44.

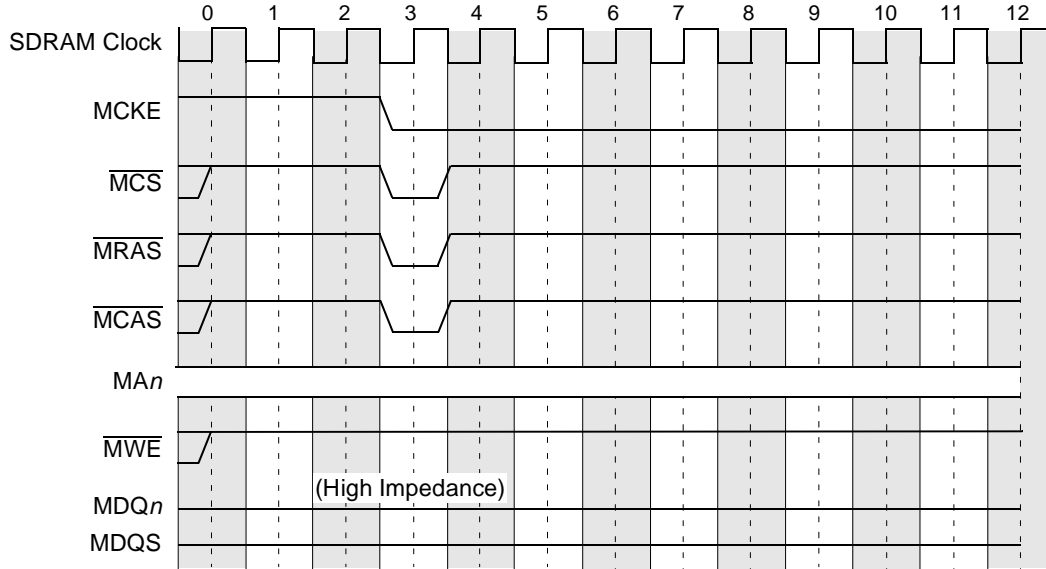


Figure 9-43. DDR SDRAM Self-Refresh Entry Timing

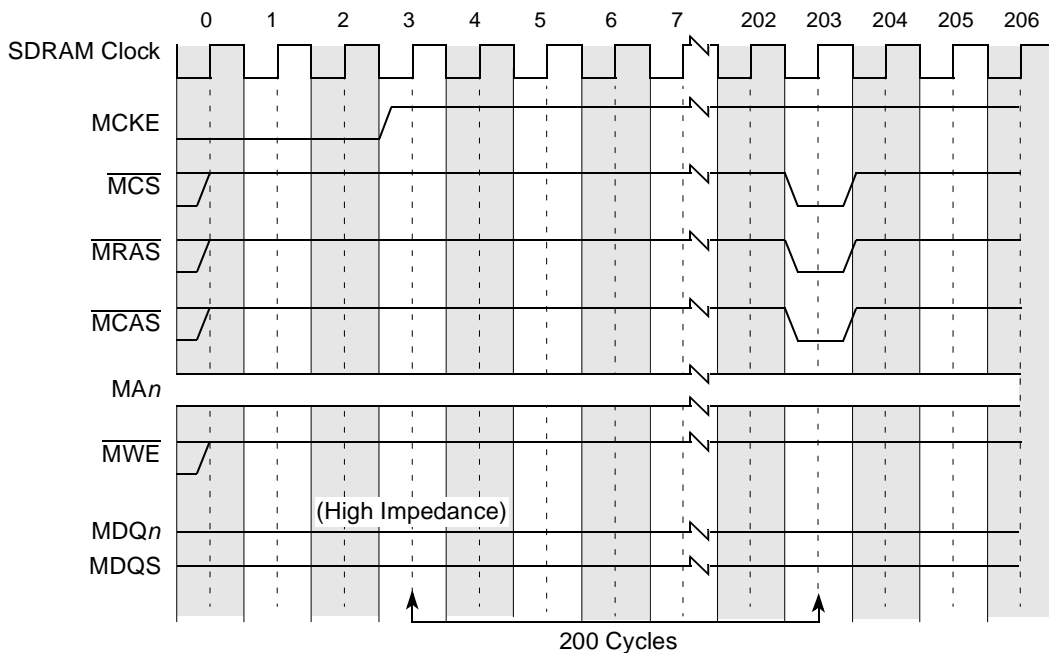


Figure 9-44. DDR SDRAM Self-Refresh Exit Timing

## 9.5.9 DDR Data Beat Ordering

Transfers to and from memory are always performed in four-beat bursts (four beats = 16 bytes when a 32-bit bus is used). For transfer sizes other than four beats, the data transfers are still operated as four-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[0:4] 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 word (4 bytes), then the second, third, and fourth beats of data are not written to DRAM (assuming a 32-bit data bus).

Table 9-44 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 9-44. Memory Controller–Data Beat Ordering**

Transfer Size	Starting Double-Word Offset	Double-Word Sequence <sup>1</sup> to/from DRAM and Queues
1 double word	0	<b>0</b> - 1 - 2 - 3
	1	<u>1</u> - 2 - 3 - 0
	2	<u>2</u> - 3 - 0 - 1
	3	<u>3</u> - 0 - 1 - 2
2 double words	0	<b>0</b> - <b>1</b> - 2 - 3
	1	<u>1</u> - <u>2</u> - 3 - 0
	2	<u>2</u> - <u>3</u> - 0 - 1
3 double words	0	<b>0</b> - <b>1</b> - <b>2</b> - 3
	1	<u>1</u> - <u>2</u> - <u>3</u> - 0

<sup>1</sup> All underlined and bolded Double-word offsets are valid for the transaction.

### 9.5.10 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 open 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 `CSn_CONFIG[AP_nEN]`.

### 9.5.11 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 9-45](#) and [Table 9-46](#).

In 32-bit mode, [Table 9-45](#) 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 9-45. DDR SDRAM ECC Syndrome Encoding**

Data Bit	Syndrome Bit							
	0	1	2	3	4	5	6	7
0	•	•						•
1	•		•					•
2	•			•				•
3	•				•			•
4	•	•				•		
5	•		•			•		
6	•			•		•		
7	•				•	•		
8	•	•					•	
9	•		•				•	
10	•			•			•	
11	•				•		•	
32			•	•				•
33			•		•			•
34	•		•		•			
35		•	•		•			
36			•	•		•		
37			•		•	•		
38	•		•		•	•		•
39		•	•		•	•		•
40			•	•			•	
41			•		•		•	
42	•		•		•		•	•
43		•	•		•		•	•

Table 9-45. DDR SDRAM ECC Syndrome Encoding (continued)

Data Bit	Syndrome Bit							
	0	1	2	3	4	5	6	7
12	•	•				•	•	•
13	•		•			•	•	•
14	•			•		•	•	•
15	•				•	•	•	•
16		•	•					•
17		•		•				•
18		•			•			•
19	•	•			•			
20		•	•			•		
21		•		•		•		
22		•			•	•		
23	•	•			•	•		•
24		•	•				•	
25		•		•			•	
26		•			•		•	
27	•	•			•		•	•
28		•	•			•	•	•
29		•		•		•	•	•
30		•			•	•	•	•
31	•	•			•	•	•	

Data Bit	Syndrome Bit							
	0	1	2	3	4	5	6	7
44			•	•		•	•	•
45			•		•	•	•	•
46	•		•		•	•	•	
47		•	•		•	•	•	
48		•				•	•	
49			•			•	•	
50				•		•	•	
51	•					•	•	
52		•				•		•
53			•			•		•
54				•		•		•
55	•					•		•
56		•					•	•
57			•				•	•
58				•			•	•
59	•						•	•
60				•	•		•	
61	•			•	•		•	•
62		•		•	•		•	•
63			•	•	•		•	•

Table 9-46. 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								•



## 9.5.12 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 9.4.1.26, “Memory Error Interrupt Enable \(ERR\\_INT\\_EN\),”](#) [Section 9.4.1.25, “Memory Error Disable \(ERR\\_DISABLE\),”](#) and [Section 9.4.1.24, “Memory Error Detect \(ERR\\_DETECT\).”](#)

Multi-bit errors can generate an MCP while single-bit errors generate a normal interrupt to the e300 core. 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 interrupt (if enabled, as described in [Section 9.4.1.25, “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 9.4.1.24, “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.

[Table 9-47](#) 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 9-47. Memory Controller Errors**

Error	Descriptions	Action	Detect Register
Single-bit ECC threshold	The number of ECC errors has reached the threshold specified in the ERR_SBE.	The error is reported through interrupt if enabled.	The error control register only logs read versus write
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.		

## 9.6 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  $\overline{MCS}_n$  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 9.4.1.2, “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 9.4.1, “Register Descriptions,”](#) for more detailed descriptions of the configuration registers. These parameters are shown in [Table 9-48](#).

**Table 9-48. Memory Interface Configuration Register Initialization Parameters**

Name	Description	Parameter	Section/page
CS <sub>n</sub> _BNDS	Chip select memory bounds	SA <sub>n</sub> EA <sub>n</sub>	<a href="#">9.4.1.1/9-10</a>
CS <sub>n</sub> _CONFIG	Chip select configuration	CS <sub>n</sub> _EN AP <sub>n</sub> _EN ODT_RD_CFG ODT_WR_CFG BA_BITS_CS <sub>n</sub> ROW_BITS_CS <sub>n</sub> COL_BITS_CS <sub>n</sub>	<a href="#">9.4.1.2/9-11</a>
TIMING_CFG_3	Extended timing parameters for fields in TIMING_CFG_1	EXT_REFREC	<a href="#">9.4.1.3/9-13</a>
TIMING_CFG_0	Timing configuration	RWT WRT RRT WWT ACT_PD_EXIT PRE_PD_EXIT ODT_PD_EXIT MRS_CYC	<a href="#">9.4.1.4/9-14</a>
TIMING_CFG_1	Timing configuration	PRETOACT ACTTOPRE ACTTORW CASLAT REFREC WRREC ACTTOACT WRTORD	<a href="#">9.4.1.5/9-16</a>
TIMING_CFG_2	Timing configuration	ADD_LAT CPO WR_LAT RD_TO_PRE WR_DATA_DELAY CKE_PLS FOUR_ACT	<a href="#">9.4.1.6/9-18</a>
DDR_SDRAM_CFG	Control configuration	SREN ECC_EN RD_EN SDRAM_TYPE DYN_PWR DBW NCAP 2T_EN BA_INTLV_CTL HSE BI	<a href="#">9.4.1.7/9-19</a>
DDR_SDRAM_CFG_2	Control configuration	DLL_RST_DIS DQS_CFG ODT_CFG NUM_PR D_INIT	<a href="#">9.4.1.8/9-22</a>
DDR_SDRAM_MODE	Mode configuration	ESDMODE SDMODE	<a href="#">9.4.1.9/9-23</a>
DDR_SDRAM_MODE_2	Mode configuration	ESDMODE2 ESDMODE3	<a href="#">9.4.1.10/9-24</a>
DDR_SDRAM_INTERVAL	Interval configuration	REFINT BSTOPRE	<a href="#">9.4.1.12/9-27</a>
DDR_DATA_INIT	Data initialization configuration register	INIT_VALUE	<a href="#">9.4.1.13/9-28</a>
DDR_SDRAM_CLK_CNTL	Clock adjust	CLK_ADJUST	<a href="#">9.4.1.14/9-28</a>
DDR_INIT_ADDR	Initialization address	INIT_ADDR	<a href="#">9.4.1.15/9-29</a>

## 9.6.1 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  $\mu$ s 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.



# Chapter 10

## 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.

### 10.1 Introduction

Figure 10-1 is a functional block diagram of the eLBC, which supports three interfaces: GPCM, FCM, and UPM controllers.

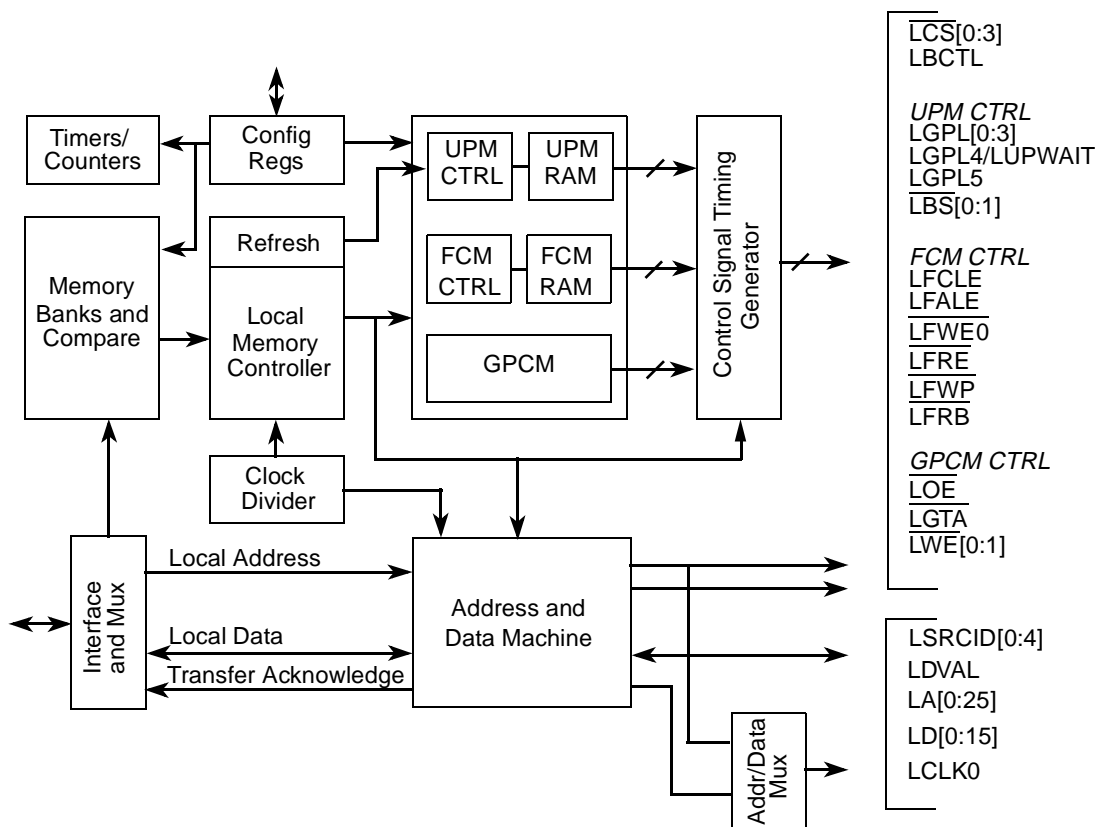


Figure 10-1. Enhanced Local Bus Controller Block Diagram

## 10.1.1 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 four 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 eLBC also includes a number of data checking and protection features such as write protection and a bus monitor to ensure that each bus cycle is terminated within a user-specified period.

## 10.1.2 Features

The eLBC main features are as follows:

- Memory controller with four memory banks
  - 32-bit address decoding with mask
  - Variable memory block sizes (32 Kbytes to 4 Gbytes in FCM mode, 32 Kbytes to 64 Mbytes in GPCM and UPM modes)
  - 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
  - Write-protection capability
- 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- and 16-bit devices
  - Minimum three-clock access to external devices
  - Two byte-write-enable signals ( $\overline{\text{LWE}}[0:1]$ )
  - 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
  - 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- and 16-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)

### 10.1.3 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 four 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 four 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.

#### 10.1.3.1 eLBC Bus Clock and Clock Ratios

The eLBC supports ratios of 2, 4, and 8 between the faster internal (system) clock and slower external bus clock (LCLK0). 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, LCLK0, to allow the clock load to be shared equally across a set of signal nets, thereby enhancing the edge rates of the bus clock.

### 10.1.3.2 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 LSRCID[0:4] whenever valid address or data is available on the eLBC external pins. When LDVAL and  $\overline{\text{LCS}}$  is asserted, valid address and data is captured from the LA and LD bus respectively.

The LSRCID[0:4] and LDVAL signals are multiplexed with other functions sharing the same external pins. Refer to [Chapter 2, “Signal Descriptions,”](#) and [Chapter 4, “Reset, Clocking, and Initialization,”](#) to learn how to enable the LSRCID/LDVAL pins.

## 10.2 External Signal Descriptions

[Table 10-1](#) contains a list of external signals related to the eLBC and summarizes their function. The table also shows the reset state of all external signals during assertion of  $\overline{\text{HRESET}}$ . For more information on the use of some of these signals as reset configuration signals on the device, see “Power on Reset Flow.”

**Table 10-1. Signal Properties—Summary**

Name	Alternate Function(s)	Mode	Descriptions	No. of Signals	I/O	Reset State (Outputs)
$\overline{\text{LCS}}[0:3]$	—	—	Chip selects 0–3	4	O	Reset_cfg
$\overline{\text{LWE}}0/$ $\overline{\text{LWE}}0/$ $\overline{\text{LBS}}0$	$\overline{\text{LWE}}0$	GPCM	Write enable 0	1	O	Reset_cfg
	$\overline{\text{LWE}}0$	FCM	Write enable	1		
	$\overline{\text{LBS}}0$	UPM	Byte (lane) select 0	1		
$\overline{\text{LWE}}1/$ $\overline{\text{LBS}}1$	$\overline{\text{LWE}}1$	GPCM	Write enable 1	1	O	Reset_cfg
	$\overline{\text{LBS}}1$	UPM	Byte (lane) select 1	1		
LGPL0/ LFCLE	LGPL0	UPM	General purpose line 0	1	O	Reset_cfg
	LFCLE	FCM	Flash command latch enable	1		
LGPL1/ LFALE	LGPL1	UPM	General purpose line 1	1	O	Reset_cfg
	LFALE	FCM	Flash address latch enable	1		
$\overline{\text{LOE}}/$ LGPL2/ $\overline{\text{LFRE}}$	$\overline{\text{LOE}}$	GPCM	Output enable	1	O	
	$\overline{\text{LFRE}}$	FCM	Flash read enable	1		
	LGPL2	UPM	General purpose line 2	1		
LGPL3/ $\overline{\text{LFWP}}$	LGPL3	UPM	General purpose line 3	1	O	Reset_cfg
	$\overline{\text{LFWP}}$	FCM	Flash write protect	1		
$\overline{\text{LGTA}}/$ $\overline{\text{LFRB}}/$ LGPL4/ LUPWAIT	$\overline{\text{LGTA}}$	GPCM	Transaction termination	1	I	High-Z
	$\overline{\text{LFRB}}$	FCM	Flash ready/busy, open-drain shared pin	1	I	
	LGPL4	UPM	General purpose line 4	1	O	
	LUPWAIT	UPM	External device wait	1	I	
LGPL5	—	UPM	General purpose line 5	1	O	Reset_cfg



Table 10-1. Signal Properties—Summary (continued)

Name	Alternate Function(s)	Mode	Descriptions	No. of Signals	I/O	Reset State (Outputs)
LBCTL	—	—	Data buffer control	1	O	
LA[0:25]	—	—	Non-multiplexed address bus	26	O	
LD[0:15]	—	—	Data bus	16	I/O	
LCLK0	—	—	Local bus clocks	1	O	Driven
LDVAL	—	eLBC debug	Local bus data valid	1	O	
LSRCID[0:4]	—	eLBC debug	Local bus source ID	5	O	

Table 10-2 shows the detailed external signal descriptions for the eLBC.

Table 10-2. Enhanced Local Bus Controller Detailed Signal Descriptions

Signal	I/O	Description
$\overline{\text{LCS}}[0:3]$	O	Chip selects. Four chip selects are provided that are mutually exclusive.
		<b>State Meaning</b> Asserted/Negated—Used to enable specific memory devices or peripherals connected to the eLBC. $\overline{\text{LCS}}[0:3]$ are provided on a per-bank basis with $\overline{\text{LCS}}_0$ corresponding to the chip select for memory bank 0, which has the memory type and attributes defined by BR0 and OR0.
$\overline{\text{LWE}}_0$ / $\overline{\text{LFW}}_0$ / $\overline{\text{LBS}}_0$ , $\overline{\text{LWE}}_1$ / $\overline{\text{LBS}}_1$	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 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.
		<b>State Meaning</b> Asserted/Negated—For GPCM operation, $\overline{\text{LWE}}[0:1]$ assert for each byte lane enabled for writing. $\overline{\text{LFW}}_0$ enables command, address, and data writes to NAND Flash EEPROMs controlled by FCM. $\overline{\text{LBS}}[0:1]$ are programmable byte-select signals in UPM mode. See Section 10.4.4.4, “RAM Array,” for programming details about $\overline{\text{LBS}}[0:1]$ .
		<b>Timing</b> Assertion/Negation—See Section 10.4.2, “General-Purpose Chip-Select Machine (GPCM),” for details regarding the timing of $\overline{\text{LWE}}[0:1]$ .
LGPL0/ LFCLE	O	General purpose line 0/FCM command latch enable.
		<b>State Meaning</b> Asserted/Negated—In UPM mode, LGPL0 is one of six general purpose signals; it is driven with a value programmed into the UPM array. In FCM mode, LFCLE enables command cycles to NAND Flash EEPROMs.
LGPL1/ LFALE	O	General-purpose line 1/FCM address latch enable.
		<b>State Meaning</b> Asserted/Negated—In UPM mode, LGPL1 is one of six general purpose signals; it is driven with a value programmed into the UPM array. In FCM mode, LFALE enables address cycles to NAND Flash EEPROMs.
$\overline{\text{LOE}}/\text{LGPL2}/$ $\overline{\text{LFRE}}$	O	GPCM output enable/General-purpose line 2/FCM read enable.
		<b>State Meaning</b> Asserted/Negated—Controls the output buffer of memory when accessing memory/devices in GPCM mode. In UPM mode, LGPL2 is one of six general purpose signals; it is driven with a value programmed into the UPM array. $\overline{\text{LFRE}}$ enables data read cycles from NAND Flash EEPROMs controlled by FCM.

Table 10-2. Enhanced Local Bus Controller Detailed Signal Descriptions (continued)

Signal	I/O	Description
LGPL3/ LFWP	O	General-purpose line 3/FCM write protect.
		<b>State Meaning</b> Asserted/Negated—In UPM mode, LGPL3 is one of six general purpose signals; it is driven with a value programmed into the UPM array. In FCM mode $\overline{\text{LFWP}}$ protects NAND Flash EEPROMs from accidental erasure and programming when $\overline{\text{LFWP}}$ is asserted low—see <a href="#">Section 10.3.1.17, “Flash Mode Register (FMR),”</a> for programming of FCM operations to control $\overline{\text{LFWP}}$ .
LGTA/LGPL4/ $\overline{\text{LFRB}}$ / LUPWAIT	I/O	GPCM transfer acknowledge/General-purpose line 4/FCM Flash ready-busy/UPM wait.
		<b>State Meaning</b> 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 $\overline{\text{LFRB}}$ to stall during long-latency read and programming operations, continuing once $\overline{\text{LFRB}}$ returns high.
LGPL5	O	General-purpose line 5
		<b>State Meaning</b> 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 $\text{OR}_n[\text{BCTLD}]$ .
		<b>State Meaning</b> Asserted/Negated—The LBCTL pin normally functions as a write/ $\overline{\text{read}}$ control for a bus transceiver connected to the LD lines. Note that an external data buffer must not drive the LD lines in conflict with the eLBC when LBCTL is high, because LBCTL remains high after reset and during address phases.
LA[0:25]	O	Nonmultiplexed address bus. All bits driven on LA[0:25] are defined for 8-bit port sizes. For 16-bit port sizes LA[25] is a don't care.
		<b>State Meaning</b> Asserted/Negated—LA is the address bus used to transmit addresses to external RAM devices. Refer to <a href="#">Section 10.5, “Initialization/Application Information,”</a> for address signal multiplexing.
LD[0:15]	I/O	Data bus. For a port size of 16 bits, LD[0:7] connect to the most-significant byte lane (at address offset 0), while LD[8:15] connect to the least-significant byte lane (at address offset 1). For a port size of 8 bits, only LD[0:7] are connected to the external RAM.
		<b>State Meaning</b> Asserted/Negated—LD is the 16-bit data bus through which external RAM devices transfer data.
LCLK0	O	Local bus clocks
		<b>State Meaning</b> Asserted/Negated—LCLK0 drive an identical bus clock signal for distributed loads.
LDVAL	O	Local bus data valid (eLBC debug mode only)
		<b>State Meaning</b> Asserted/Negated—For a read, LDVAL asserts for one bus cycle in the cycle immediately preceding the sampling of read data on LD. For a write, LDVAL asserts for one bus cycle during the final cycle for which the current write data on LD is valid. During burst transfers, LDVAL asserts for each data beat.
		<b>Timing</b> Assertion/Negation—Valid only while the eLBC is in system debug mode. In debug mode, LDVAL asserts when the eLBC generates a data transfer acknowledge.

**Table 10-2. Enhanced Local Bus Controller Detailed Signal Descriptions (continued)**

Signal	I/O	Description
LSRCID[0:4]	O	Local bus source ID (eLBC debug mode only). In debug mode, all LSRCID[0:4] pins are driven high unless LSRCID[0:4] is driving a debug source ID for identifying the internal system device controlling the eLBC.
		<b>State Meaning</b>

### 10.3 Memory Map/Register Definition

Table 10-3 shows the memory mapped registers of the eLBC. Undefined 4-byte address spaces within offset 0x000–0xFFF are reserved.

**Table 10-3. 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	10.3.1.1/10-9
0x008	BR1—Base register 1	R/W	0x0000_0000	10.3.1.1/10-9
0x010	BR2—Base register 2	R/W	0x0000_0000	10.3.1.1/10-9
0x018	BR3—Base register 3	R/W	0x0000_0000	10.3.1.1/10-9
0x020–0x038	Reserved	—	—	—
0x004	OR0—Options register 0	R/W	0x0000_0FF7	10.3.1.2/10-10
0x00C	OR1—Options register 1	R/W	0x0000_0000	10.3.1.2/10-10
0x014	OR2—Options register 2	R/W	0x0000_0000	10.3.1.2/10-10
0x01C	OR3—Options register 3	R/W	0x0000_0000	10.3.1.2/10-10
0x024–0x064	Reserved	—	—	—
0x068	MAR—UPM address register	R/W	0x0000_0000	10.3.1.3/10-18
0x06C	Reserved	—	—	—
0x070	MAMR—UPMA mode register	R/W	0x0000_0000	10.3.1.4/10-19
0x074	MBMR—UPMB mode register	R/W	0x0000_0000	10.3.1.4/10-19
0x078	MCMR—UPMC mode register	R/W	0x0000_0000	10.3.1.4/10-19
0x07C–0x080	Reserved	—	—	—
0x084	MRTPR—Memory refresh timer prescaler register	R/W	0x0000_0000	10.3.1.5/10-21
0x088	MDR—UPM/FCM data register	R/W	0x0000_0000	10.3.1.6/10-21
0x08C	Reserved	—	—	—
0x090	LSOR—Special operation initiation register	R/W	0x0000_0000	10.3.1.7/10-22

Table 10-3. Enhanced Local Bus Controller Registers (continued)

Enhanced Local Bus Controller—Block Base Address 0x0_5000				
Offset	Register	Access	Reset	Section/Page
0x094–0x09C	Reserved	—	—	—
0x0A0	LURT—UPM refresh timer	R/W	0x0000_0000	<a href="#">10.3.1.4/10-19</a>
0x0A4–0x0AC	Reserved	—	—	—
0x0B0	LTESR—Transfer error status register	w1c	0x0000_0000	<a href="#">10.3.1.9/10-24</a>
0x0B4	LTEDR—Transfer error disable register	R/W	0x0000_0000	<a href="#">10.3.1.10/10-26</a>
0x0B8	LTEIR—Transfer error interrupt register	R/W	0x0000_0000	<a href="#">10.3.1.11/10-27</a>
0x0BC	LTEATR—Transfer error attributes register	R/W	0x0000_0000	<a href="#">10.3.1.12/10-28</a>
0x0C0	LTEAR—Transfer error address register	R/W	0x0000_0000	<a href="#">10.3.1.13/10-29</a>
0x0C4	LTECCR—Transfer error ECC register	w1c	0x0000_0000	<a href="#">10.3.1.14/10-29</a>
0x0C8–0x0CC	Reserved	—	—	—
0x0D0	LBCR—Configuration register	R/W	0x0004_0000	<a href="#">10.3.1.15/10-30</a>
0x0D4	LCRR—Clock ratio register	R/W	0x8000_0008	<a href="#">10.3.1.16/10-31</a>
0x0D8–0x0DC	Reserved	—	—	—
0x0E0	FMR—Flash mode register	R/W	0x0000_0n00	<a href="#">10.3.1.17/10-32</a>
0x0E4	FIR—Flash instruction register	R/W	0x0000_0000	<a href="#">10.3.1.18/10-34</a>
0x0E8	FCR—Flash command register	R/W	0x0000_0000	<a href="#">10.3.1.19/10-35</a>
0x0EC	FBAR—Flash block address register	R/W	0x0000_0000	<a href="#">10.3.1.20/10-36</a>
0x0F0	FPAR—Flash page address register	R/W	0x0000_0000	<a href="#">10.3.1.21/10-36</a>
0x0F4	FBCR—Flash byte count register	R/W	0x0000_0000	<a href="#">10.3.1.22/10-38</a>
0x0F8–0x0FC	Reserved	—	—	—
0x100	FECC0—Flash ECC block 0 register	R	0x0000_0000	<a href="#">10.3.1.23/10-38</a>
0x104	FECC1—Flash ECC block 1 register	R	0x0000_0000	<a href="#">10.3.1.23/10-38</a>
0x108	FECC2—Flash ECC block 2 register	R	0x0000_0000	<a href="#">10.3.1.23/10-38</a>
0x10C	FECC3—Flash ECC block 3 register	R	0x0000_0000	<a href="#">10.3.1.23/10-38</a>

### 10.3.1 Register Descriptions

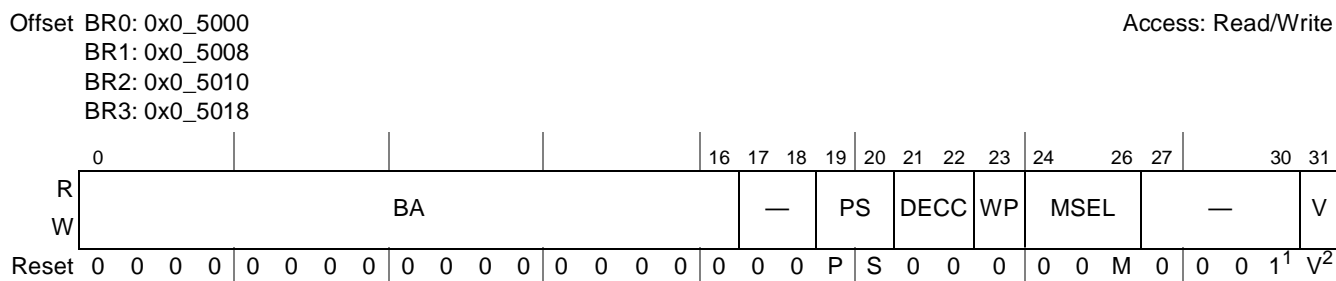
This section provides a detailed description of the eLBC configuration, status, and control registers with detailed bit and field descriptions.

Address offsets in the eLBC address range that are not defined in [Table 10-3](#) 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.

### 10.3.1.1 Base Registers (BR0–BR3)

The base registers ( $BR_n$ ), shown in [Figure 10-2](#), 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,  $BR_0[V]$  is set,  $BR_1[V]$ – $BR_3[V]$  are cleared, and the value of  $BR_0[PS]$  reflects the initial port size configured by the boot ROM location field of the reset configuration word.



**Figure 10-2. Base Registers ( $BR_n$ )**

<sup>1</sup> Writes to this bit are ignored.

<sup>2</sup>  $BR_0$  has its valid bit (V) set for  $RCWH[ROMLOC] = LBC$ . Thus bank 0 is valid with the port size (PS) configured from  $RCWH[ROMLOC]$  as loaded during reset.  $M = 0$  for MSEL of GPCM, 1 for MSEL of FCM at boot. All other base registers have all bits cleared to zero during reset.

[Table 10-4](#) describes  $BR_n$  fields.

**Table 10-4.  $BR_n$  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 $BR_0$ , PS is configured from the field in reset configuration word as loaded during reset. For all other banks the value is reset to 00 (port size not defined). 00 Reserved 01 8-bit (supported for GPCM, UPM, FCM) 10 16-bit (supported for GPCM, UPM) 11 Reserved

Table 10-4. BR<sub>n</sub> Field Descriptions (continued)

Bits	Name	Description
21–22	DECC	Specifies the method for data error checking. 00 Data error checking disabled. No ECC generation for FCM. 01 ECC checking is enabled, but ECC generation is disabled, for FCM on full-page transfers. 10 ECC checking and generation are enabled for FCM on full-page transfers. 11 Reserved
23	WP	Write protect. 0 Read and write accesses are allowed. 1 Only read accesses are allowed. The memory controller does not assert $\overline{LCSn}$ 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. 000 GPCM (possible reset value) 001 FCM (possible reset value) 010 Reserved 011 Reserved 100 UPMA 101 UPMB 110 UPMC 111 Reserved
27–30	—	Reserved
31	V	Valid bit. Indicates that the contents of the BR <sub>n</sub> and OR <sub>n</sub> pair are valid. $\overline{LCSn}$ 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. 0 This bank is invalid. 1 This bank is valid.

### 10.3.1.2 Option Registers (OR0–OR3)

The OR<sub>n</sub> registers define the sizes of memory banks and access attributes. The OR<sub>n</sub> attribute bits support the following three modes of operation as defined by BR<sub>n</sub>[MSEL]:

- GPCM mode
- FCM mode
- UPM mode

The OR<sub>n</sub> 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 10-5 shows the reset values for OR0.

Table 10-5. 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

### 10.3.1.2.1 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 10-6 shows memory bank sizes from 32 Kbytes to 4 Gbytes. Memory block sizes vary from 32 Kbytes to 4 Gbytes in FCM mode, and 32 Kbytes to 64 Mbytes in GPCM and UPM modes.

**Table 10-6. Memory Bank Sizes in Relation to Address Mask**

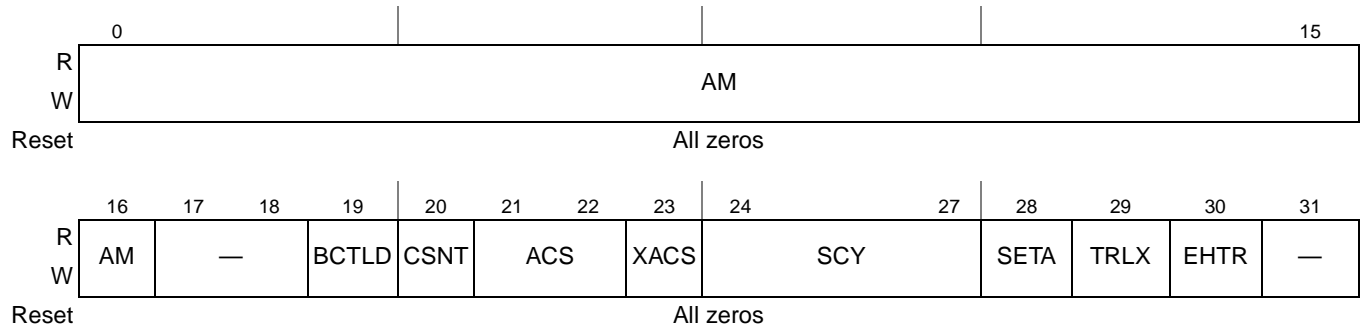
AM	Memory Bank Size
0000_0000_0000_0000_0	4 Gbytes
1000_0000_0000_0000_0	2 Gbytes
1100_0000_0000_0000_0	1 Gbyte
1110_0000_0000_0000_0	512 Mbytes
1111_0000_0000_0000_0	256 Mbytes
1111_1000_0000_0000_0	128 Mbytes
1111_1100_0000_0000_0	64 Mbytes
1111_1110_0000_0000_0	32 Mbytes
1111_1111_0000_0000_0	16 Mbytes
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
1111_1111_1111_1111_0	64 Kbytes
1111_1111_1111_1111_1	32 Kbytes

### 10.3.1.2.2 Option Registers (OR<sub>n</sub>)—GPCM Mode

Figure 10-3 shows the bit fields for OR<sub>n</sub> when the corresponding BR<sub>n</sub>[MSEL] selects the GPCM machine.

Offset OR0: 0x0\_5004  
 OR1: 0x0\_500c  
 OR2: 0x0\_5014  
 OR3: 0x0\_501c

Access: Read/Write



<sup>1</sup> Refer to Table 10-5 for the OR0 reset value. All other option registers have all bits cleared.

**Figure 10-3. Option Registers (OR<sub>n</sub>) in GPCM Mode**

Table 10-7 describes OR<sub>n</sub> fields for GPCM mode.

**Table 10-7. OR<sub>n</sub>—GPCM Field Descriptions**

Bits	Name	Description												
0–16	AM	GPCM address mask. Masks corresponding BR <sub>n</sub> 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. 0 Corresponding address bits are masked and therefore don't care for address checking. 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. 0 LBCTL is asserted upon access to the current memory bank. 1 LBCTL is not asserted upon access to the current memory bank.												
20	CSNT	Chip select negation time. Determines when $\overline{LCSn}$ and $\overline{LWE}$ are negated during an external memory write access handled by the GPCM, provided that ACS ≠ 00 (when ACS = 00, only $\overline{LWE}$ is affected by the setting of CSNT). This helps meet address/data hold times for slow memories and peripherals. 0 $\overline{LCSn}$ and $\overline{LWE}$ are negated normally. 1 $\overline{LCSn}$ and $\overline{LWE}$ are negated earlier depending on the value of LCRR[CLKDIV].												
<table border="1"> <thead> <tr> <th>LCRR [CLKDIV]</th> <th>CSNT</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>x</td> <td>0</td> <td><math>\overline{LCSn}</math> and <math>\overline{LWE}</math> are negated normally.</td> </tr> <tr> <td>2</td> <td>1</td> <td><math>\overline{LCSn}</math> and <math>\overline{LWE}</math> are negated normally.</td> </tr> <tr> <td>4 or 8</td> <td>1</td> <td><math>\overline{LCSn}</math> and <math>\overline{LWE}</math> are negated one quarter bus clock cycle earlier.</td> </tr> </tbody> </table>			LCRR [CLKDIV]	CSNT	Meaning	x	0	$\overline{LCSn}$ and $\overline{LWE}$ are negated normally.	2	1	$\overline{LCSn}$ and $\overline{LWE}$ are negated normally.	4 or 8	1	$\overline{LCSn}$ and $\overline{LWE}$ are negated one quarter bus clock cycle earlier.
LCRR [CLKDIV]	CSNT	Meaning												
x	0	$\overline{LCSn}$ and $\overline{LWE}$ are negated normally.												
2	1	$\overline{LCSn}$ and $\overline{LWE}$ are negated normally.												
4 or 8	1	$\overline{LCSn}$ and $\overline{LWE}$ are negated one quarter bus clock cycle earlier.												



Table 10-7. OR $n$ —GPCM Field Descriptions (continued)

Bits	Name	Description																		
21–22	ACS	<p>Address to chip-select setup. Determines the delay of the <math>\overline{\text{LCS}}n</math> assertion relative to the address change when the external memory access is handled by the GPCM. At system reset, OR0[ACS] = 11.</p> <table border="1"> <thead> <tr> <th>LCRR [CLKDIV]</th> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td rowspan="2">x</td> <td>00</td> <td><math>\overline{\text{LCS}}n</math> 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 rowspan="2">2</td> <td>10</td> <td><math>\overline{\text{LCS}}n</math> is output one half bus clock cycle after the address lines.</td> </tr> <tr> <td>11</td> <td><math>\overline{\text{LCS}}n</math> is output one half bus clock cycle after the address lines.</td> </tr> <tr> <td rowspan="2">4 or 8</td> <td>10</td> <td><math>\overline{\text{LCS}}n</math> is output one quarter bus clock cycle after the address lines.</td> </tr> <tr> <td>11</td> <td><math>\overline{\text{LCS}}n</math> is output one half bus clock cycle after the address lines.</td> </tr> </tbody> </table>	LCRR [CLKDIV]	Value	Meaning	x	00	$\overline{\text{LCS}}n$ is output at the same time as the address lines. Note that this overrides the value of CSNT such that CSNT = 0.	01	Reserved.	2	10	$\overline{\text{LCS}}n$ is output one half bus clock cycle after the address lines.	11	$\overline{\text{LCS}}n$ is output one half bus clock cycle after the address lines.	4 or 8	10	$\overline{\text{LCS}}n$ is output one quarter bus clock cycle after the address lines.	11	$\overline{\text{LCS}}n$ is output one half bus clock cycle after the address lines.
LCRR [CLKDIV]	Value	Meaning																		
x	00	$\overline{\text{LCS}}n$ is output at the same time as the address lines. Note that this overrides the value of CSNT such that CSNT = 0.																		
	01	Reserved.																		
2	10	$\overline{\text{LCS}}n$ is output one half bus clock cycle after the address lines.																		
	11	$\overline{\text{LCS}}n$ is output one half bus clock cycle after the address lines.																		
4 or 8	10	$\overline{\text{LCS}}n$ is output one quarter bus clock cycle after the address lines.																		
	11	$\overline{\text{LCS}}n$ is output one half bus clock cycle after the address lines.																		
23	XACS	<p>Extra address to chip-select setup. Setting this bit increases the delay of the <math>\overline{\text{LCS}}n</math> assertion relative to the address change when the external memory access is handled by the GPCM. After a system reset, OR0[XACS] = 1.</p> <p>0 Address to chip-select setup is determined by ORx[ACS] and LCRR[CLKDIV].  1 Address to chip-select setup is extended (see Table 10-32 and Table 10-33).</p>																		
24–27	SCY	<p>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.</p> <p>0000 No wait states  0001 1 bus clock cycle wait state  ...  1111 15 bus clock cycle wait states</p>																		
28	SETA	<p>External address termination.</p> <p>0 Access is terminated internally by the memory controller unless the external device asserts <math>\overline{\text{LGT}}A</math> earlier to terminate the access.  1 Access is terminated externally by asserting the <math>\overline{\text{LGT}}A</math> external pin. (Only <math>\overline{\text{LGT}}A</math> can terminate the access).</p>																		
29	TRLX	<p>Timing relaxed. Modifies the settings of timing parameters for slow memories or peripherals.</p> <p>0 Normal timing is generated by the GPCM.  1 Relaxed timing on the following parameters:</p> <ul style="list-style-type: none"> <li>• Adds an additional cycle between the address and control signals (only if ACS is not equal to 00).</li> <li>• Doubles the number of wait states specified by SCY, providing up to 30 wait states.</li> <li>• Works in conjunction with EHTR to extend hold time on read accesses.</li> <li>• <math>\overline{\text{LCS}}n</math> (only if ACS is not equal to 00) and <math>\overline{\text{LWE}}</math> signals are negated one cycle earlier during writes.</li> </ul>																		

Table 10-7. OR<sub>n</sub>—GPCM Field Descriptions (continued)

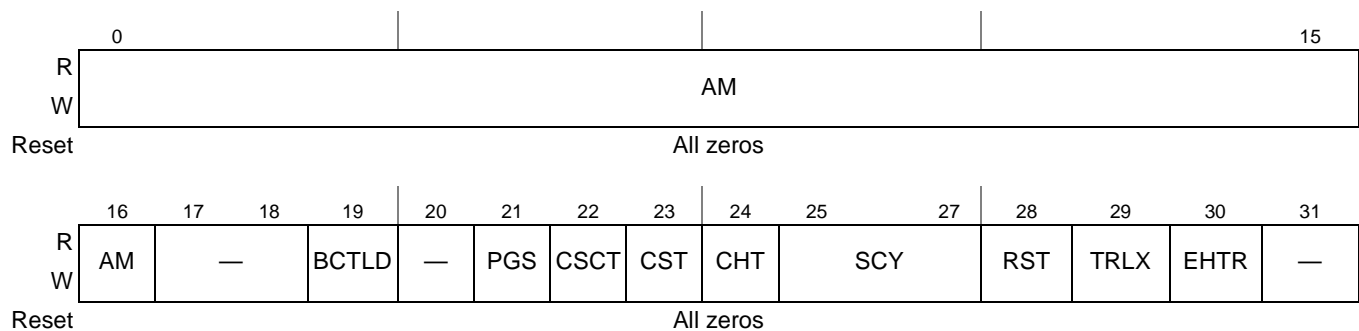
Bits	Name	Description															
30	EHTR	Extended hold time on read accesses. Indicates with TRLX how many cycles are inserted between a read access from the current bank and the next access. <table border="1" style="margin: 10px auto;"> <thead> <tr> <th>TRLX</th> <th>EHTR</th> <th>Meaning</th> </tr> </thead> <tbody> <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> </tbody> </table>	TRLX	EHTR	Meaning	0	0	The memory controller generates normal timing. No additional cycles are inserted.	0	1	1 idle clock cycle is inserted.	1	0	4 idle clock cycles are inserted.	1	1	8 idle clock cycles are inserted.
TRLX	EHTR	Meaning															
0	0	The memory controller generates normal timing. No additional cycles are inserted.															
0	1	1 idle clock cycle is inserted.															
1	0	4 idle clock cycles are inserted.															
1	1	8 idle clock cycles are inserted.															
31	—	Reserved.															

### 10.3.1.2.3 Option Registers (OR<sub>n</sub>)—FCM Mode

Figure 10-4 shows the bit fields for OR<sub>n</sub> when the corresponding BR<sub>n</sub>[MSEL] selects the FCM machine.

Offset OR0: 0x0\_5004  
OR1: 0x0\_500c  
OR2: 0x0\_5014  
OR3: 0x0\_501c

Access: Read/Write



<sup>1</sup> Refer to Table 10-5 for the OR0 reset value. All other option registers have all bits cleared.

Figure 10-4. Option Registers (OR<sub>n</sub>) in FCM Mode

Table 10-8 describes OR<sub>n</sub> fields for FCM mode.

Table 10-8. OR<sub>n</sub>—FCM Field Descriptions

Bits	Name	Description
0–16	AM	FCM address mask. Masks corresponding BR <sub>n</sub> 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. 0 Corresponding address bits are masked. 1 Corresponding address bits are used in the comparison between base and transaction addresses.
17–18	—	Reserved

Table 10-8. OR $n$ —FCM Field Descriptions (continued)

Bits	Name	Description															
19	BCTLD	Buffer control disable. Disables assertion of LBCTL during access to the current memory bank. 0 LBCTL is asserted upon access to the current memory bank. 1 LBCTL is not asserted upon access to the current memory bank.															
20	—	Reserved															
21	PGS	NAND Flash EEPROM page size, buffer size, and block size. 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. 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.															
22	CSCT	Chip select to command time. Determines how far in advance $\overline{LCSn}$ 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 border="1" data-bbox="376 699 1430 940"> <thead> <tr> <th>TRLX</th> <th>CSCT</th> <th>Meaning</th> </tr> </thead> <tbody> <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> </tbody> </table>	TRLX	CSCT	Meaning	0	0	The chip-select is asserted 1 clock cycle before any command.	0	1	The chip-select is asserted 4 clock cycles before any command.	1	0	The chip-select is asserted 2 clock cycles before any command.	1	1	The chip-select is asserted 8 clock cycles before any command.
TRLX	CSCT	Meaning															
0	0	The chip-select is asserted 1 clock cycle before any command.															
0	1	The chip-select is asserted 4 clock cycles before any command.															
1	0	The chip-select is asserted 2 clock cycles before any command.															
1	1	The chip-select is asserted 8 clock cycles before any command.															
23	CST	Command setup time. Determines the delay of $\overline{LFWEO}$ assertion relative to the command, address, or data change when the external memory access is handled by the FCM. <table border="1" data-bbox="376 1045 1430 1339"> <thead> <tr> <th>TRLX</th> <th>CST</th> <th>Meaning</th> </tr> </thead> <tbody> <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> </tbody> </table>	TRLX	CST	Meaning	0	0	The write-enable is asserted coincident with any command.	0	1	The write-enable is asserted 0.25 clock cycles after any command, address, or data.	1	0	The write-enable is asserted 0.5 clock cycles after any command, address, or data.	1	1	The write-enable is asserted 1 clock cycle after any command, address, or data.
TRLX	CST	Meaning															
0	0	The write-enable is asserted coincident with any command.															
0	1	The write-enable is asserted 0.25 clock cycles after any command, address, or data.															
1	0	The write-enable is asserted 0.5 clock cycles after any command, address, or data.															
1	1	The write-enable is asserted 1 clock cycle after any command, address, or data.															
24	CHT	Command hold time. Determines the $\overline{LFWEO}$ negation prior to the command, address, or data change when the external memory access is handled by the FCM. <table border="1" data-bbox="376 1444 1430 1801"> <thead> <tr> <th>TRLX</th> <th>CHT</th> <th>Meaning</th> </tr> </thead> <tbody> <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> </tbody> </table>	TRLX	CHT	Meaning	0	0	The write-enable is negated 0.5 clock cycles before any command, address, or data change.	0	1	The write-enable is negated 1 clock cycle before any command, address, or data change.	1	0	The write-enable is negated 1.5 clock cycles before any command, address, or data change.	1	1	The write-enable is negated 2 clock cycles before any command, address, or data change.
TRLX	CHT	Meaning															
0	0	The write-enable is negated 0.5 clock cycles before any command, address, or data change.															
0	1	The write-enable is negated 1 clock cycle before any command, address, or data change.															
1	0	The write-enable is negated 1.5 clock cycles before any command, address, or data change.															
1	1	The write-enable is negated 2 clock cycles before any command, address, or data change.															

Table 10-8. OR<sub>n</sub>—FCM Field Descriptions (continued)

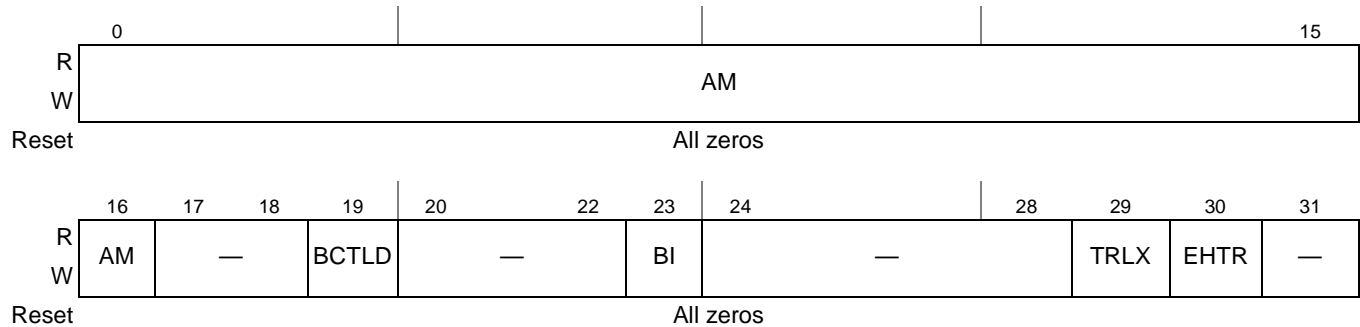
Bits	Name	Description															
25–27	SCY	<p>Cycle length in bus clocks. Determines:</p> <ul style="list-style-type: none"> <li>the number of wait states inserted in command, address, or data transfer bus cycles, when the FCM handles the external memory access. Thus it is the main parameter for determining cycle length. The total cycle length depends on other timing attribute settings.</li> <li>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 <math>4 \times (2 + \text{SCY})</math> clock cycles (TRLX = 0) or <math>8 \times (2 + \text{SCY})</math> clock cycles (TRLX = 1) is inserted between the last write and the first data transfer to/from NAND Flash devices.</li> <li>the delay between a command write and the first sample point of the RDY/<math>\overline{\text{BSY}}</math> pin (connected to <math>\overline{\text{LFRB}}</math>). <math>\overline{\text{LFRB}}</math> is not sampled until <math>8 \times (2 + \text{SCY})</math> clock cycles (TRLX = 0) or <math>16 \times (2 + \text{SCY})</math> clock cycles (TRLX = 1) have elapsed following the command.</li> </ul> <p>000 No extra wait states  001 1 bus clock cycle wait state  ...  111 7 bus clock cycle wait states</p>															
28	RST	<p>Read setup time. Determines the delay of <math>\overline{\text{LFRB}}</math> assertion relative to sampling of read data when the external memory access is handled by the FCM.</p> <table border="1"> <thead> <tr> <th>TRLX</th> <th>RST</th> <th>Meaning</th> </tr> </thead> <tbody> <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> </tbody> </table>	TRLX	RST	Meaning	0	0	The read-enable is asserted 0.75 clock cycles prior to any wait states.	0	1	The read-enable is asserted 1 clock cycle prior to any wait states.	1	0	The read-enable is asserted 0.5 clock cycles prior to any wait states.	1	1	The read-enable is asserted 1 clock cycle prior to any wait states.
TRLX	RST	Meaning															
0	0	The read-enable is asserted 0.75 clock cycles prior to any wait states.															
0	1	The read-enable is asserted 1 clock cycle prior to any wait states.															
1	0	The read-enable is asserted 0.5 clock cycles prior to any wait states.															
1	1	The read-enable is asserted 1 clock cycle prior to any wait states.															
29	TRLX	<p>Timing relaxed. Modifies the settings of timing parameters for slow memories.</p> <p>0 Normal timing is generated by the FCM.  1 Relaxed timing on the following parameters:</p> <ul style="list-style-type: none"> <li>Doubles the number of clock cycles between <math>\overline{\text{LCS}}_n</math> assertion and commands.</li> <li>Doubles the number of wait states specified by SCY, providing up to 14 wait states.</li> <li>Works in conjunction with CST and RST to extend command/address/data setup times.</li> <li>Adds one clock cycle to the command/address/data hold times.</li> <li>Works in conjunction with CBT to extend the wait time for read/busy status sampling by 16 clock cycles.</li> <li>Works in conjunction with EHTR to double hold time on read accesses.</li> </ul>															
30	EHTR	<p>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.</p> <table border="1"> <thead> <tr> <th>TRLX</th> <th>EHTR</th> <th>Meaning</th> </tr> </thead> <tbody> <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> </tbody> </table>	TRLX	EHTR	Meaning	0	0	1 idle clock cycle is inserted.	0	1	2 idle clock cycles are inserted.	1	0	4 idle clock cycles are inserted.	1	1	8 idle clock cycles are inserted.
TRLX	EHTR	Meaning															
0	0	1 idle clock cycle is inserted.															
0	1	2 idle clock cycles are inserted.															
1	0	4 idle clock cycles are inserted.															
1	1	8 idle clock cycles are inserted.															
31	—	Reserved															

### 10.3.1.2.4 Option Registers (OR<sub>*n*</sub>)—UPM Mode

Figure 10-5 shows the bit fields for OR<sub>*n*</sub> when the corresponding BR<sub>*n*</sub>[MSEL] selects a UPM machine.

Offset OR0: 0x0\_5004  
 OR1: 0x0\_500c  
 OR2: 0x0\_5014  
 OR3: 0x0\_501c

Access: Read/Write



<sup>1</sup> Refer to Table 10-5 for the OR0 reset value. All other option registers have all bits cleared.

**Figure 10-5. Option Registers (OR<sub>*n*</sub>) in UPM Mode**

Table 10-9 describes BR<sub>*n*</sub> fields for UPM mode.

**Table 10-9. OR<sub>*n*</sub>—UPM Field Descriptions**

Bits	Name	Description
0–16	AM	UPM address mask. Masks corresponding BR <sub><i>n</i></sub> 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. 0 Corresponding address bits are masked. 1 The corresponding address bits are used in the comparison with address pins.
17–18	—	Reserved
19	BCTLD	Buffer control disable. Disables assertion of LBCTL during access to the current memory bank. 0 LBCTL is asserted upon access to the current memory bank. 1 LBCTL is not asserted upon access to the current memory bank.
20–22	—	Reserved
23	BI	Burst inhibit. Indicates if this memory bank supports burst accesses. 0 The bank supports burst accesses. 1 The bank does not support burst accesses. The selected UPM executes burst accesses as a series of single accesses.
24–28	—	Reserved
29	TR LX	Timing relaxed. Works in conjunction with EHTR to extend hold time on read accesses.

Table 10-9. OR<sub>n</sub>—UPM 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. <table border="1" data-bbox="354 352 1433 594"> <thead> <tr> <th>TRLX</th> <th>EHTR</th> <th>Meaning</th> </tr> </thead> <tbody> <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> </tbody> </table>	TRLX	EHTR	Meaning	0	0	The memory controller generates normal timing. No additional cycles are inserted.	0	1	1 idle clock cycle is inserted.	1	0	4 idle clock cycles are inserted.	1	1	8 idle clock cycles are inserted.
TRLX	EHTR	Meaning															
0	0	The memory controller generates normal timing. No additional cycles are inserted.															
0	1	1 idle clock cycle is inserted.															
1	0	4 idle clock cycles are inserted.															
1	1	8 idle clock cycles are inserted.															
31	—	Reserved															

### 10.3.1.3 UPM Memory Address Register (MAR)

Figure 10-6 shows the fields of the UPM memory address register (MAR).



Figure 10-6. UPM Memory Address Register (MAR)

Table 10-10 describes the MAR fields.

Table 10-10. MAR Field Descriptions

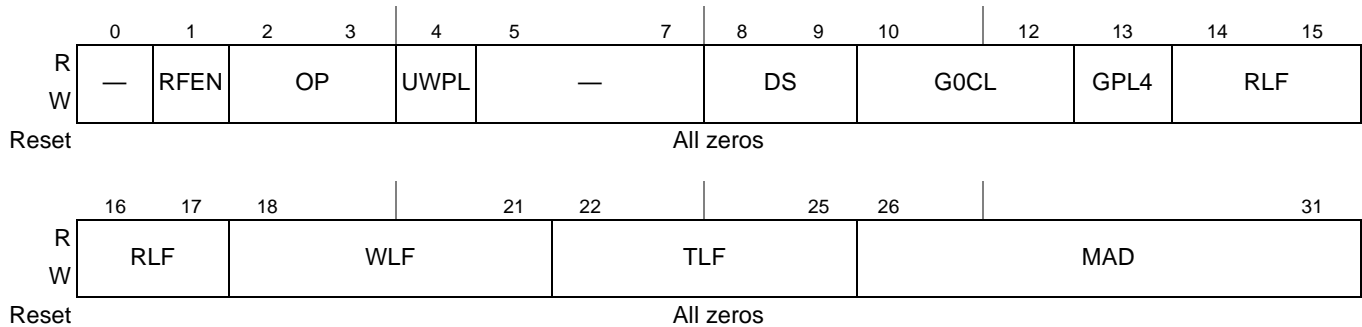
Bits	Name	Description
0–31	A	Address that can be output to the address signals.

### 10.3.1.4 UPM Mode Registers (MxMR)

The UPM machine mode registers (MAMR, MBMR and MCMR), shown in [Figure 10-7](#), contain the configuration for the three UPMs.

Offset MAMR: 0x0\_5070  
 MBMR: 0x0\_5074  
 MCMR: 0x0\_5078

Access: Read/Write



**Figure 10-7. UPM Mode Registers (MxMR)**

[Table 10-11](#) describes UPM mode fields.

**Table 10-11. MxMR 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. 0 Refresh services are not required 1 Refresh services are required
2–3	OP	Command opcode. Determines the command executed by the UPM <sub>n</sub> when a memory access hits a UPM assigned bank. 00 Normal operation 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. 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. 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. 0 LUPWAIT is active high. 1 LUPWAIT is active low.
5–7	—	Reserved

Table 10-11. MxMR Field Descriptions (continued)

Bits	Name	Description														
8–9	DS	<p>Disable timer period. Guarantees a minimum time between accesses to the same memory bank controlled by UPM<math>n</math>. The disable timer is turned on by the TODT bit in the RAM array word, and when expired, the UPM<math>n</math> allows the machine access to handle a memory pattern to the same bank. Accesses to a different bank by the same UPM<math>n</math> 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.</p> <p>00 1-bus clock cycle disable period  01 2-bus clock cycle disable period  10 3-bus clock cycle disable period  11 4-bus clock cycle disable period</p>														
10–12	GOCL	<p>General line 0 control. Determines which logical address line can be output to the LGPL0 pin when the UPM<math>n</math> is selected to control the memory access.</p> <p>000 A12  001 A11  010 A10  011 A9  100 A8  101 A7  110 A6  111 A5</p>														
13	GPL4	<p>LGPL4 output line disable. Determines how the LGPL4/LUPWAIT pin is controlled by the corresponding bits in the UPM<math>n</math> array. See <a href="#">Table 10-40 on page 10-75</a>.</p> <table border="1" data-bbox="375 976 1170 1167"> <thead> <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> </thead> <tbody> <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> </tbody> </table>	Value	LGPL4/LUPWAIT Pin Function	Interpretation of UPM Word Bits		G4T1/DLT3	G4T3/WAEN	0	LGPL4 (output)	G4T1	G4T3	1	LUPWAIT (input)	DLT3	WAEN
Value	LGPL4/LUPWAIT Pin Function	Interpretation of UPM Word Bits														
		G4T1/DLT3	G4T3/WAEN													
0	LGPL4 (output)	G4T1	G4T3													
1	LUPWAIT (input)	DLT3	WAEN													
14–17	RLF	<p>Read loop field. Determines the number of times a loop defined in the UPM<math>n</math> will be executed for a burst- or single-beat read pattern or when MxMR[OP] = 11 (RUN command)</p> <p>0000 16  0001 1  0010 2  0011 3  ...  1110 14  1111 15</p>														
18–21	WLF	<p>Write loop field. Determines the number of times a loop defined in the UPM<math>n</math> will be executed for a burst- or single-beat write pattern.</p> <p>0000 16  0001 1  0010 2  0011 3  ...  1110 14  1111 15</p>														



Table 10-11. MxMR Field Descriptions (continued)

Bits	Name	Description
22–25	TLF	Refresh loop field. Determines the number of times a loop defined in the UPM <sub>n</sub> will be executed for a refresh service pattern. 0000 16 0001 1 0010 2 0011 3 ... 1110 14 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 <sub>n</sub> .

### 10.3.1.5 Memory Refresh Timer Prescaler Register (MRTPR)

The refresh timer prescaler register (MRTPR), shown in Figure 10-8, is used to divide the system clock to provide the UPM refresh timers clock.

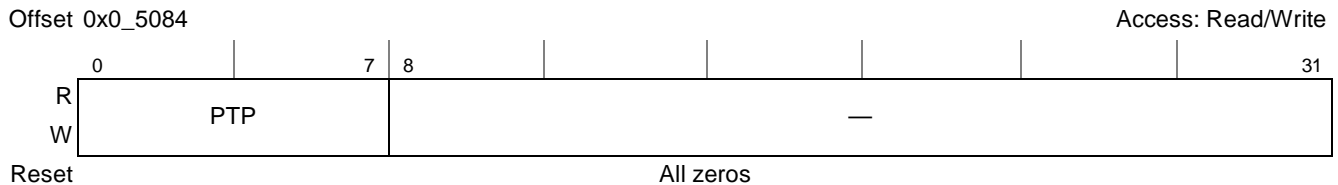


Figure 10-8. Memory Refresh Timer Prescaler Register (MRTPR)

Table 10-12 describes MRTPR fields.

Table 10-12. 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

### 10.3.1.6 UPM/FCM Data Register (MDR)

The memory data register (MDR), shown in Figure 10-9 and Figure 10-10, 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.

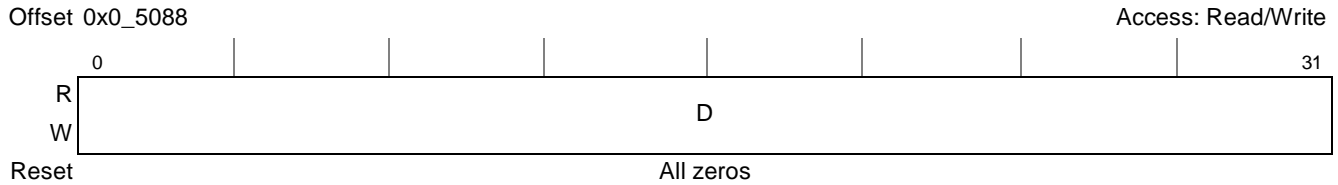


Figure 10-9. UPM Data Register in UPM Mode (MDR)

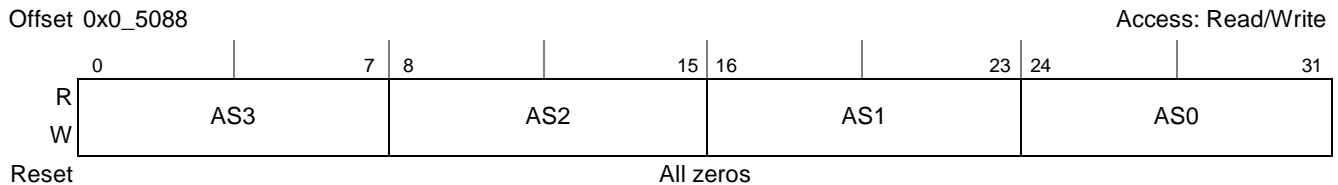


Figure 10-10. FCM Data Register in FCM Mode (MDR)

Table 10-13 describes MDR[D].

Table 10-13. MDR Field Description

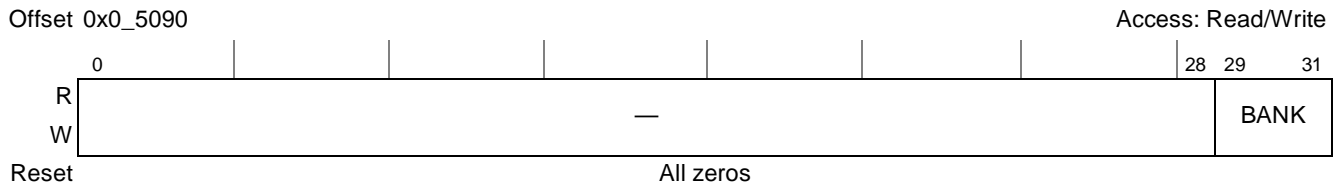
Bits	Name	Description
0–31	D	In UPM mode, D is the data to be read or written into the RAM array when a write or read command is supplied to the UPM (MxMR[OP] = 01 or MxMR[OP] = 10).
0–7	AS3	In FCM mode, AS3 is the fourth byte of address sent by a custom address write operation, or the fourth byte of 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.

### 10.3.1.7 Special Operation Initiation Register (LSOR)

The special operation initiation register (LSOR), shown in Figure 10-11, 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 MxMR[OP], see Section 10.3.1.4, “UPM Mode Registers (MxMR).” FCM special operation modes are set in FMR[OP], see Section 10.3.1.17, “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 10.4.4.2.1, “UPM Programming Example \(Two Sequential Writes to the RAM Array\).”](#)



**Figure 10-11. Special Operation Initiation Register (LSOR)**

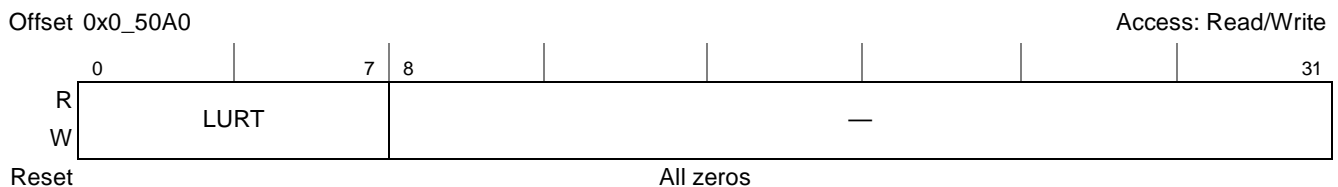
[Table 10-14](#) describes LSOR.

**Table 10-14. 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. 000 Bank 0 is triggered for special operation ... 011 Bank 3 is triggered for special operation 100–111 Reserved

### 10.3.1.8 UPM Refresh Timer (LURT)

The UPM refresh timer (LURT), shown in [Figure 10-12](#), 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 10-12. UPM Refresh Timer (LURT)**

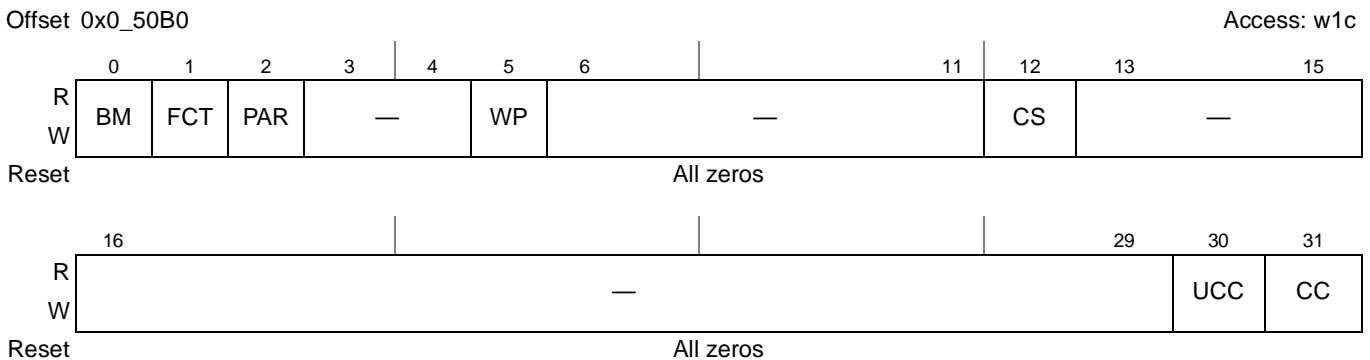
Table 10-15 describes LURT fields.

**Table 10-15. LURT Field Descriptions**

Bits	Name	Description
0–7	LURT	<p>UPM refresh timer period. Determines, along with the timer prescaler (MRTPR), the timer period according to the following equation:</p> $\text{TimerPeriod} = \frac{\text{LURT}}{\left(\frac{\text{Fsystemclock}}{\text{MRTPR}[\text{PTP}]}\right)}$ <p>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. Note that the reset value (0x00) sets the maximum period to 256 x MRTPR[PTP] system clock cycles.</p>
8–31	—	Reserved

### 10.3.1.9 Transfer Error Status Register (LTESR)

The transfer error status register (LTESR) indicates the cause of an error or event. LTESR, shown in Figure 10-13, 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.



**Figure 10-13. Transfer Error Status Register (LTESR)**

Table 10-16 describes LTESR fields.

**Table 10-16. LTESR Field Descriptions**

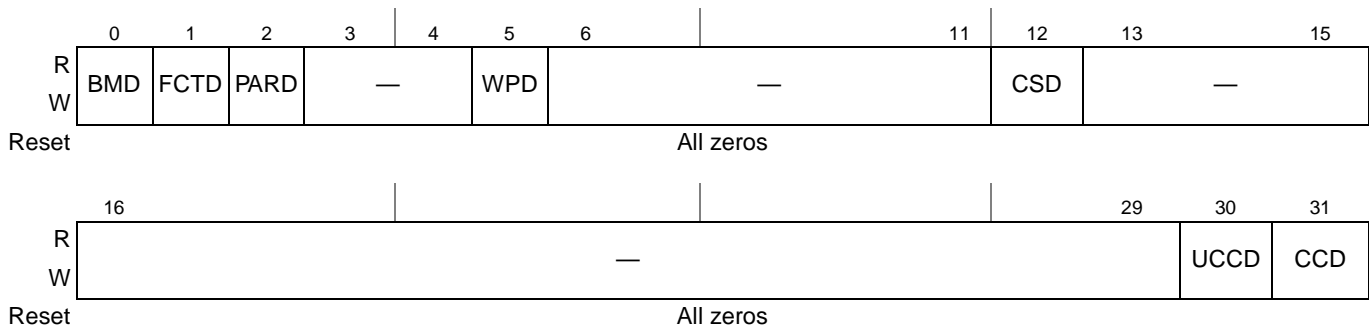
Bits	Name	Description
0	BM	Bus monitor time-out 0 No local bus monitor time-out occurred. 1 Local bus monitor time-out occurred. No data beat was acknowledged on the bus within LBCR[BMT] x LBCR[BMTPS] bus clock cycles from the start of a transaction.
1	FCT	FCM command time-out 0 No FCM command time-out occurred. 1 A CW0, CW1, CW2, or CW3 command issued to FCM timed-out with respect to the timer configured by FMR[CWTO].
2	PAR	ECC error for FCM mode 0 No local bus ECC error 1 Uncorrectable ECC error (FCM). LTEATR[PB] indicates the block that caused the error and LTEATR[BNK] indicates which memory controller bank was accessed.
3–4	—	Reserved
5	WP	Write protect error 0 No write protect error occurred. 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).
6–11	—	Reserved
12	CS	Chip select error 0 No chip select error occurred. 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 0 No UPM Run pattern operation in progress, or operation pending. 1 UPM Run pattern operation has completed, allowing software to continue processing of results.
31	CC	FCM command completion event 0 No FCM operation in progress, or operation pending. 1 FCM operation has completed, allowing software to continue processing of results.

### 10.3.1.10 Transfer Error Check Disable Register (LTEDR)

The transfer error check disable register (LTEDR), shown in Figure 10-14, 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



**Figure 10-14. Transfer Error Check Disable Register (LTEDR)**

Table 10-17 describes LTEDR fields.

**Table 10-17. LTEDR Field Descriptions**

Bits	Name	Description
0	BMD	Bus monitor disable 0 Bus monitor is enabled. 1 Bus monitor is disabled, but internal bus time-outs can still occur.
1	FCTD	FCM command time-out disable 0 FCM command timer is enabled. 1 FCM command time-out is disabled, but internal FCM command timer can terminate command waits.
2	PARD	ECC error checking disabled. 0 ECC error checking is enabled. 1 ECC error checking is disabled.
3–4	—	Reserved
5	WPD	Write protect error checking disable. 0 Write protect error checking is enabled. 1 Write protect error checking is disabled.
6–11	—	Reserved
12	CSD	Chip select error checking disable. 0 Chip select error checking is enabled. 1 Chip select error checking is disabled.
13–29	—	Reserved
30	UCCD	UPM Run pattern command completion checking disable. 0 UPM Run pattern command completion checking is enabled. 1 UPM Run pattern command completion checking is disabled.
31	CCD	FCM command completion checking disable. 0 Command completion checking is enabled. 1 Command completion checking is disabled.

### 10.3.1.11 Transfer Error Interrupt Enable Register (LTEIR)

The transfer error interrupt enable register (LTEIR), shown in [Figure 10-15](#), 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.



**Figure 10-15. Transfer Error Interrupt Enable Register (LTEIR)**

[Table 10-18](#) describes LTEIR fields.

**Table 10-18. LTEIR Field Descriptions**

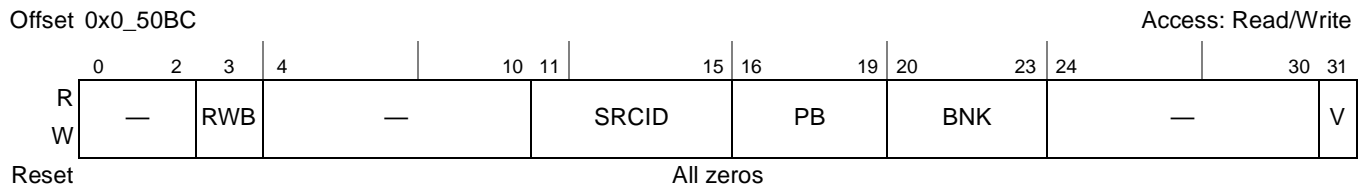
Bits	Name	Description
0	BMI	Bus monitor error interrupt enable. 0 Bus monitor error reporting is disabled. 1 Bus monitor error reporting is enabled.
1	FCTI	FCM command time-out interrupt enable. 0 FCM command time-out error reporting is disabled. 1 FCM command time-out error reporting is enabled.
2	PARI	ECC error interrupt enable. 0 ECC error reporting is disabled. 1 ECC error reporting is enabled.
3–4	—	Reserved
5	WPI	Write protect error interrupt enable. 0 Write protect error reporting is disabled. 1 Write protect error reporting is enabled.
6–11	—	Reserved
12	CSI	Chip select error interrupt enable. 0 Chip select error reporting is disabled. 1 Chip select error reporting is enabled.
13–29	—	Reserved

**Table 10-18. LTEIR Field Descriptions (continued)**

Bits	Name	Description
30	UCCI	UPM Run pattern command completion Event interrupt enable. 0 UPM Run pattern command completion reporting is disabled. 1 UPM Run pattern command completion reporting is enabled.
31	CCI	FCM command completion Event interrupt enable. 0 Command completion reporting is disabled. 1 Command completion reporting is enabled.

### 10.3.1.12 Transfer Error Attributes Register (LTEATR)

The transfer error attributes register (LTEATR) captures source attributes of an error/event. [Figure 10-16](#) 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.

**Figure 10-16. Transfer Error Attributes Register (LTEATR)**

[Table 10-19](#) describes LTEATR fields.

**Table 10-19. LTEATR Field Descriptions**

Bits	Name	Description
0–2	—	Reserved
3	RWB	Transaction type for the error: 0 The transaction for the error was a write transaction. 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. For more information, see <a href="#">Table 6-7</a> in <a href="#">Section 6.2.6, “Arbiter Event Attributes Register (AEATR).”</a>
16–19	PB	Error on block for FCM. 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).
20–23	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.
24–30	—	Reserved
31	V	Error attribute capture is valid. Indicates that the captured error information is valid. 0 Captured error attributes and address are not valid. 1 Captured error attributes and address are valid.



### 10.3.1.13 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 10-17.

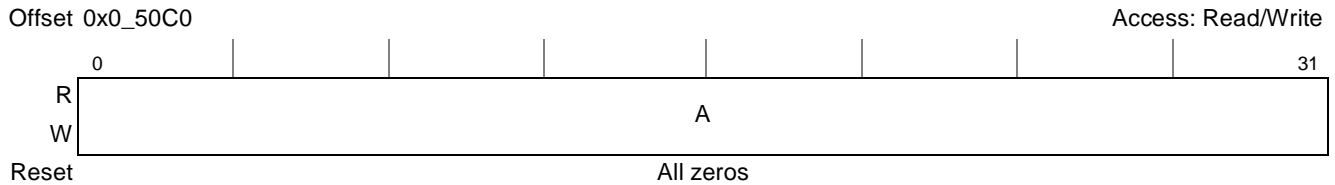


Figure 10-17. Transfer Error Address Register (LTEAR)

Table 10-20 describes LTEAR fields.

Table 10-20. 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.

### 10.3.1.14 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 10-18, 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].

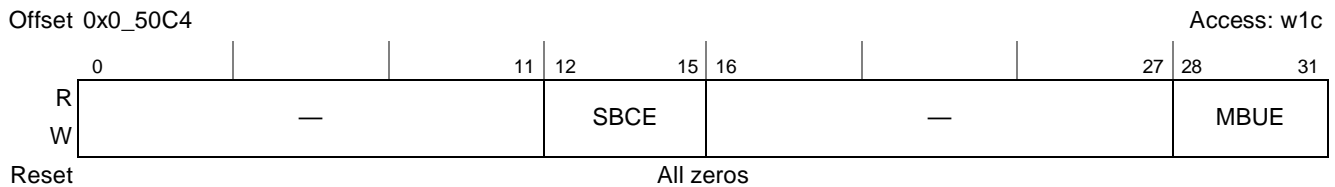


Figure 10-18. Transfer Error ECC Register (LTECCR)

Table 10-21. LTECCR Field Descriptions

Bits	Name	Description
0–11	—	Reserved
12–15	SBCE	Single bit correctable error 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 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).

### 10.3.1.15 Local Bus Configuration Register (LBCR)

The local bus configuration register (LBCR) is shown in Figure 10-19.

Offset 0x0\_50D0

Access: Read/Write

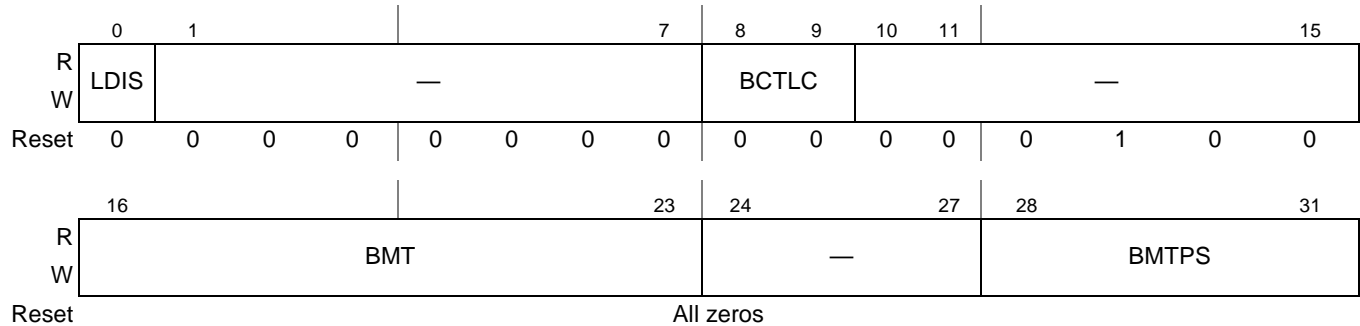


Figure 10-19. Local Bus Configuration Register

Table 10-22 describes LBCR fields.

Table 10-22. LBCR Field Descriptions

Bits	Name	Description
0	LDIS	Local bus disable 0 Local bus is enabled. 1 Local bus is disabled. No internal transactions will be acknowledged.
1–7	—	Reserved
8–9	BCTLC	Defines the use of LBCTL 00 LBCTL is used as $W/\overline{R}$ control for GPCM or UPM accesses (buffer control). 01 LBCTL is used as $\overline{LOE}$ for GPCM accesses only. 10 LBCTL is used as $\overline{LWE}$ for GPCM accesses only. 11 Reserved.
10	—	Reserved
11–15	—	Reserved. <b>Note:</b> Reads to bit 13 return 1. During writes, it is recommended to write 1 to bit 13.
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: bus cycles = BMT × PS, where PS is set according to LBCR[BMTPS]. The value of BMT × PS must not be less than 40 bus cycles for reliable operation.

Table 10-22. 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. 0000 PS = 8 0001 PS = 16 0010 PS = 32 0011 PS = 64 0100 PS = 128 0101 PS = 256 0110 PS = 512 0111 PS = 1024 1000 PS = 2048 1001 PS = 4096 1010 PS = 8192 1011 PS = 16,384 1100 PS = 32,768 1101 PS = 65,536 1110 PS = 131,072 1111 PS = 262,144

### 10.3.1.16 Clock Ratio Register (LCRR)

The clock ratio register, shown in Figure 10-20, 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.

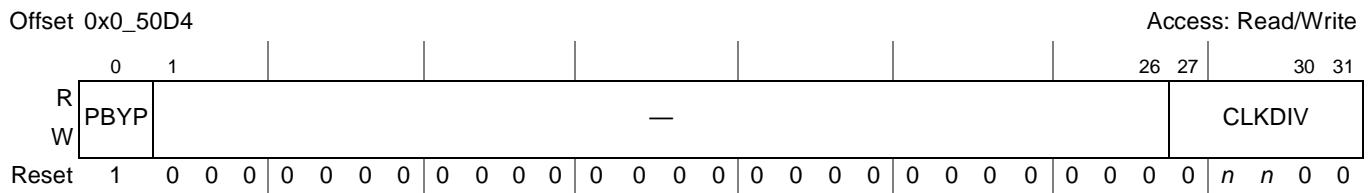


Figure 10-20. Clock Ratio Register (LCRR)

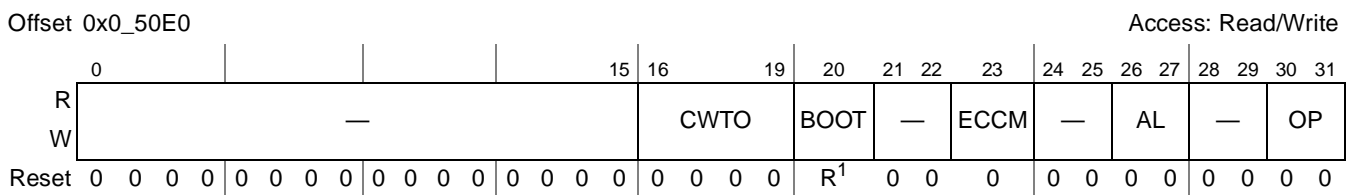
Table 10-23 describes LCRR fields.

**Table 10-23. LCRR Field Descriptions**

Bits	Name	Description
0	PBYP	PLL bypass. This bit should be set when using low bus clock frequencies (66 MHz or lower). When in PLL bypass mode, incoming data is captured in the middle of the bus clock cycle. 0 The PLL is enabled. 1 The PLL is bypassed.
1–26	—	Reserved Although bit 14 and 15 are reserved, they can be still programmed with the following to provide additional delay to LCLK: 00 4 01 1 10 2 11 3
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 <code>csb_clk</code> or twice <code>csb_clk</code> (if <code>RCWL[LBIUCM]</code> is set). Only the values shown below are allowed. <b>Note:</b> It is critical that no transactions are being executed via the local bus while <code>CLKDIV</code> is being modified. As such, prior to modification, the user must ensure that code is not executing out of the local bus. Once <code>LCRR[CLKDIV]</code> is written, the register should be read, and then an <code>isync</code> should be executed.  00000–00001 Reserved 00010 2 00011 Reserved 00100 4 00101–00111 Reserved 01000 8 01001–11111 Reserved

### 10.3.1.17 Flash Mode Register (FMR)

The local bus Flash mode register (FMR), shown in Figure 10-21, controls global operation of the FCM.



**Figure 10-21. Flash Mode Register**

<sup>1</sup> Bit R (field BOOT) is set if power-on-reset configuration selects FCM as the boot ROM target.

Table 10-24 describes FMR fields.

**Table 10-24. 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: 0000 256 cycles of LCLK0 0001 512 cycles of LCLK0 0010 1024 cycles of LCLK0 0011 2048 cycles of LCLK0 0100 4096 cycles of LCLK0 0101 8192 cycles of LCLK0 0110 16,384 cycles of LCLK0 0111 32,768 cycles of LCLK0 1000 65,536 cycles of LCLK0 1001 131,072 cycles of LCLK0 1010 262,144 cycles of LCLK0 1011 524,288 cycles of LCLK0 1100 1,048,576 cycles of LCLK0 1101 2,097,152 cycles of LCLK0 1110 4,194,304 cycles of LCLK0 1111 8,388,608 cycles of LCLK0
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. 0 FCM is operating in normal functional mode, with an 8 Kbyte FCM buffer RAM. 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 BRn[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. 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. 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 10-24. 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). 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 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 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 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. 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. 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. 10 Execute the command sequence contained in FIR, but with write protection enabled (pin $\overline{LFWP}$ asserted low) so that all Flash blocks are protected from accidental erasure and reprogramming. 11 Execute the command sequence contained in FIR, but permit the single block identified by FBAR[BLK] to be erased or reprogrammed, with pin $\overline{LFWP}$ remaining high during the access.

### 10.3.1.18 Flash Instruction Register (FIR)

The local bus Flash instruction register (FIR), shown in Figure 10-22, 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

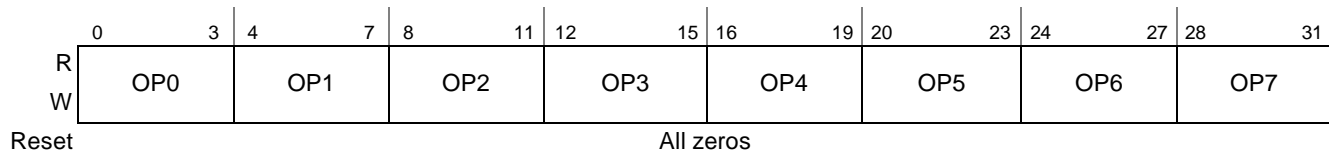


Figure 10-22. Flash Instruction Register

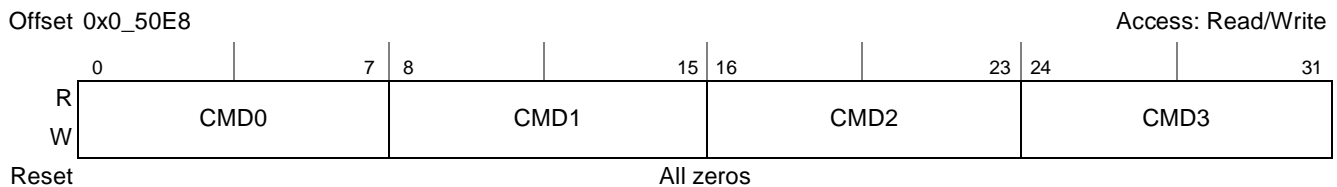
Table 10-25 describes FIR fields.

**Table 10-25. FIR Field Descriptions**

Bits	Name	Description
0–3	OP0	FCM operation codes. OP0 is executed first, followed by OP1, through to OP7.
4–7	OP1	0000 NOP—No-operation and end of operation sequence
8–11	OP2	0001 CA—Issue current column address as set in FPAR, with length set by ORx[PGS] 0010 PA—Issue current block+page address as set in FBAR and FPAR, with length set by FMR[AL] 0011 UA—Issue user-defined address byte from next AS field in MDR
12–15	OP3	0100 CM0—Issue command from FCR[CMD0]
16–19	OP4	0101 CM1—Issue command from FCR[CMD1] 0110 CM2—Issue command from FCR[CMD2]
20–23	OP5	0111 CM3—Issue command from FCR[CMD3]
24–27	OP6	1000 WB—Write FBCR bytes of data from current FCM buffer to Flash device 1001 WS—Write one byte (8b port) of data from next AS field of MDR to Flash device
28–31	OP7	1010 RB—Read FBCR bytes of data from Flash device into current FCM RAM buffer 1011 RS—Read one byte (8b port) of data from Flash device into next AS field of MDR 1100 CW0—Wait for $\overline{\text{LFRB}}$ to return high or time-out, then issue command from FCR[CMD0] 1101 CW1—Wait for $\overline{\text{LFRB}}$ to return high or time-out, then issue command from FCR[CMD1] 1110 RBW—Wait for $\overline{\text{LFRB}}$ to return high or time-out, then read FBCR bytes of data from Flash device into current FCM RAM buffer 1111 RSW—Wait for $\overline{\text{LFRB}}$ to return high or time-out, then read one byte (8b port) of data from Flash device into next AS field of MDR

### 10.3.1.19 Flash Command Register (FCR)

The local bus Flash command register (FCR), shown in Figure 10-23, 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.



**Figure 10-23. Flash Command Register**

Table 10-26 describes FCR fields.

**Table 10-26. FCR Field Descriptions**

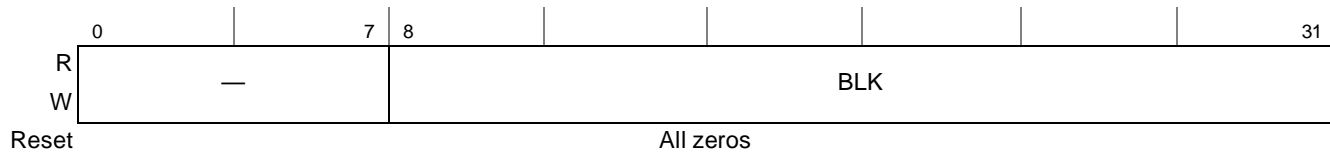
Bits	Name	Description
0–7	CMD0	General purpose FCM Flash command byte 0. Opcodes in FIR that issue command index 0 write CMD0 to the NAND Flash command/data bus.
8–15	CMD1	General purpose FCM Flash command byte 1. Opcodes in FIR that issue command index 1 write CMD1 to the NAND Flash command/data bus.
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.

### 10.3.1.20 Flash Block Address Register (FBAR)

The local bus Flash block address register (FBAR), shown in [Figure 10-24](#), locates the NAND Flash block index for the page currently accessed.

Offset 0x0\_50EC

Access: Read/Write



**Figure 10-24. Flash Block Address Register**

[Table 10-27](#) describes FBAR fields.

**Table 10-27. FBAR Field Descriptions**

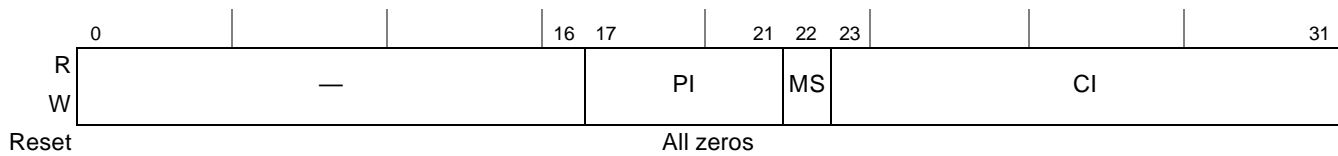
Bits	Name	Description
0–7	—	Reserved
8–31	BLK	Flash block address. The size of the NAND Flash, as configured in OR <sub>n</sub> [PGS] and FMR[AL], determines the number of bits of BLK that are issued to the EEPROM during block address phases.

### 10.3.1.21 Flash Page Address Register (FPAR)

The local bus Flash page address register (FPAR), shown in [Figure 10-25](#) and [Figure 10-26](#), locates the current NAND Flash page in both the external NAND Flash device and FCM buffer RAM.

Offset 0x0\_50F0

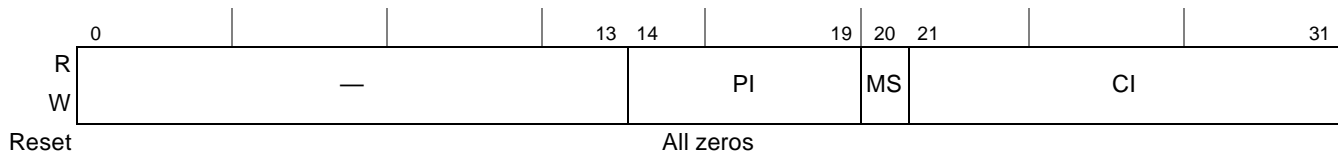
Access: Read/Write



**Figure 10-25. Flash Page Address Register, Small Page Device (OR<sub>x</sub>[PGS] = 0)**

Offset 0x0\_50F0

Access: Read/Write



**Figure 10-26. Flash Page Address Register, Large Page Device (OR<sub>x</sub>[PGS] = 1)**



Table 10-28 describes FPAR fields for small page devices.

**Table 10-28. 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. The 3 LSBs of PI index one of the eight 1 Kbyte buffers in the FCM buffer RAM as follows: 000 The page is transferred to/from FCM buffer 0, address offsets 0x0000–0x03FF 001 The page is transferred to/from FCM buffer 1, address offsets 0x0400–0x07FF 010 The page is transferred to/from FCM buffer 2, address offsets 0x0800–0x0BFF 011 The page is transferred to/from FCM buffer 3, address offsets 0x0C00–0x0FFF 100 The page is transferred to/from FCM buffer 4, address offsets 0x1000–0x13FF 101 The page is transferred to/from FCM buffer 5, address offsets 0x1400–0x17FF 110 The page is transferred to/from FCM buffer 6, address offsets 0x1800–0x1BFF 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. 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. 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.

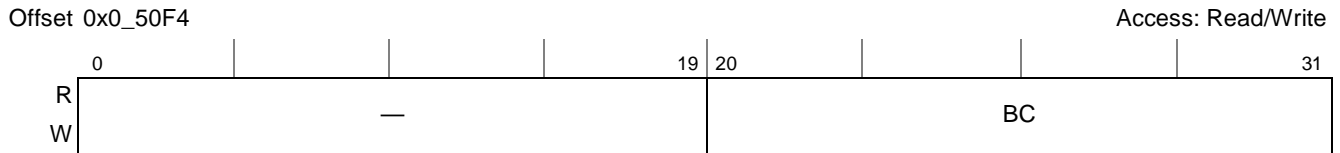
Table 10-29 describes FPAR fields for large page devices.

**Table 10-29. 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. The LSB of PI indexes one of the two 4 Kbyte buffers in the FCM buffer RAM as follows: 0 The page is transferred to/from FCM buffer 0, address offsets 0x0000–0x0FFF 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. 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. 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.

### 10.3.1.22 Flash Byte Count Register (FBCR)

The local bus Flash byte count register (FBCR), shown in [Figure 10-27](#), defines the size of FCM block transfers for reads and writes to the NAND Flash EEPROM.



**Figure 10-27. Flash Byte Count Register**

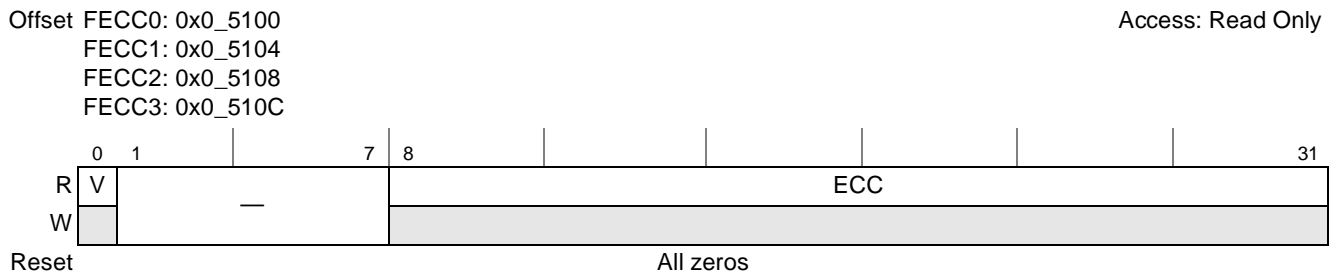
[Table 10-30](#) describes FBCR fields.

**Table 10-30. 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. 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. 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.

### 10.3.1.23 Flash ECC Block $n$ Register (FECC0–FECC3)

The local bus flash ECC block $n$  register (FECC $n$ ), shown in [Figure 10-28](#), 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 10-28. Flash ECC Block $n$  Register (FECC0–FECC3)**

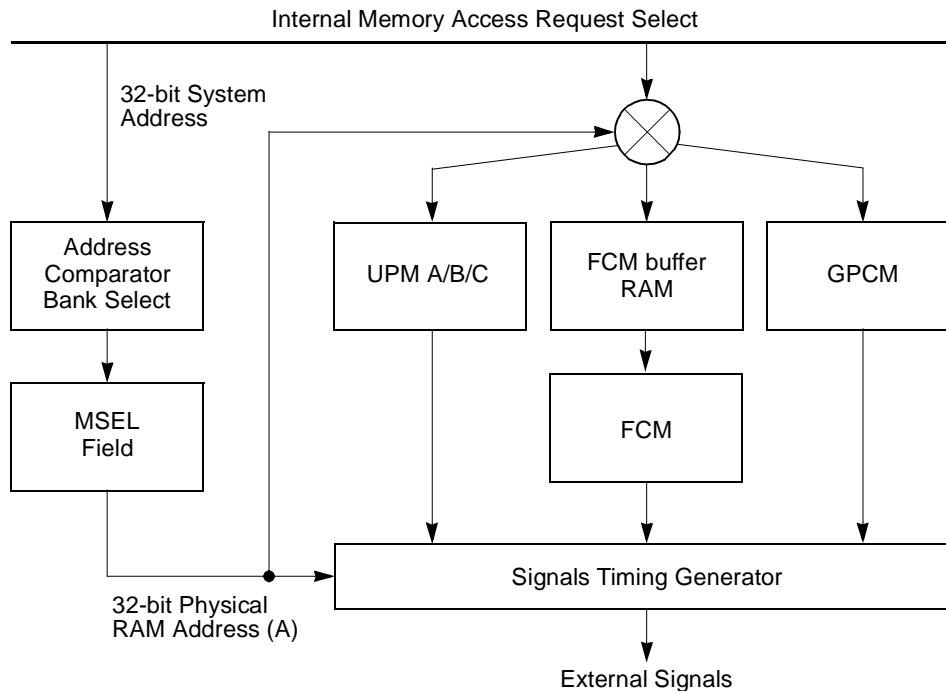
Table 10-31. FECC $n$  Field Descriptions

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 $n^{\text{th}}$ 512 bytes of a page in case of large page or for $(4k + n)^{\text{th}}$ 512 byte page for small page where $k = 0, 1, 2, \dots$ . It stores calculated ECC value during writes/reads.

## 10.4 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 10-29. 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 10-29](#). 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.

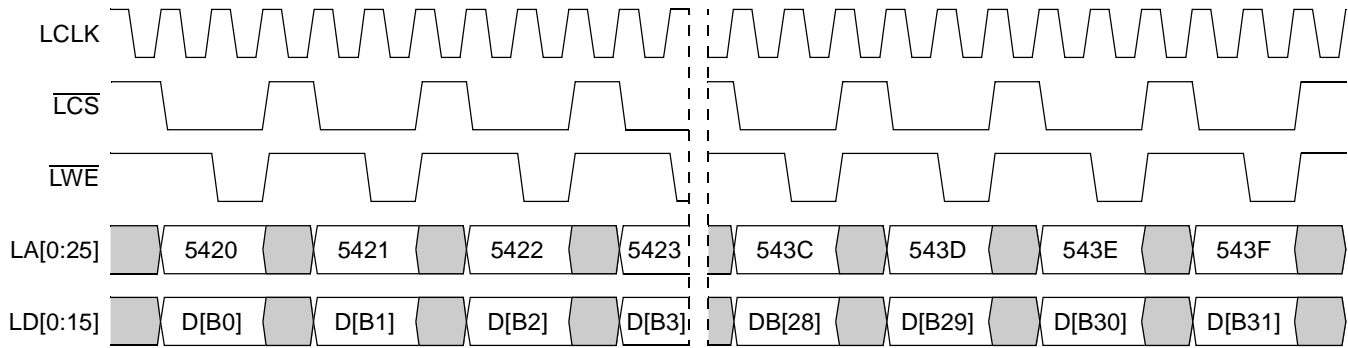
## 10.4.1 Basic Architecture

The following subsections describe the basic architecture of the eLBC.

### 10.4.1.1 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).

To illustrate how a large transaction is handled by the eLBC, [Figure 10-30](#) shows eLBC signals for the GPCM performing a 32-byte write starting at address 0x5420.



Note: All address and signal values are shown in hexadecimal.  
D[Bk] = k<sup>th</sup> of 32 data bytes.

Figure 10-30. Example of 8-Bit GPCM Writing 32 Bytes to Address 0x5420 (LCRR[PBYP] = 0)

### 10.4.1.2 Data Transfer Acknowledge (TA)

The three memory controllers in the eLBC generate an internal transfer acknowledge signal, TA, to allow data on LD 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 LDVAL 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 10-31 shows TA (internal), and  $\overline{LCS}_n$ .

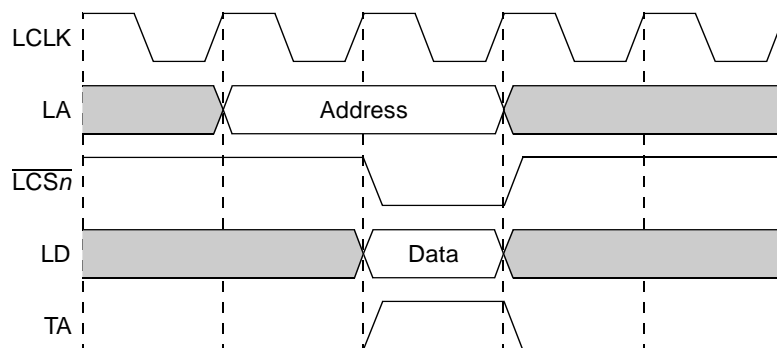


Figure 10-31. Basic eLBC Bus Cycle with TA, and  $\overline{LCS}_n$

### 10.4.1.3 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  $OR_n[BCTL D]$ . LBCTL can be further configured by  $LBCR[BCTL C]$  to act as an extra  $\overline{LWE}$  or an extra  $\overline{LOE}$  signal when in GPCM mode.

If LBCTL is configured as a data buffer control ( $LBCR[BCTL C] = 00$ ), the signal is asserted (high) on the rising edge of the bus clock on the first cycle of the memory controller operation. 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 assertion of  $\overline{LCS}$  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 to allow a whole bus cycle for the bus to turn around before the next address is driven.

#### 10.4.1.4 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 ( $LBCR[BMT] \times LBCR[BMTPS]$ ) 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 (that is, 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 10.4.4.1.4, “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.

### 10.4.2 General-Purpose Chip-Select Machine (GPCM)

The GPCM allows a minimal glue logic and flexible interface to SRAM, EPROM, FEPRAM, ROM devices, and external peripherals. The GPCM contains two basic configuration register groups— $BR_n$  and  $OR_n$ .

Figure 10-32 shows a simple connection between an 8-bit port size SRAM device and the eLBC in GPCM mode. Byte-write enable signals ( $\overline{LWE}$ ) are available for each byte written to memory. Also, the output enable signal ( $\overline{LOE}$ ) is provided to minimize external glue logic. On system reset, a global (boot) chip-select is available that provides a boot ROM chip-select ( $\overline{LCS0}$ ) prior to the system being fully configured.

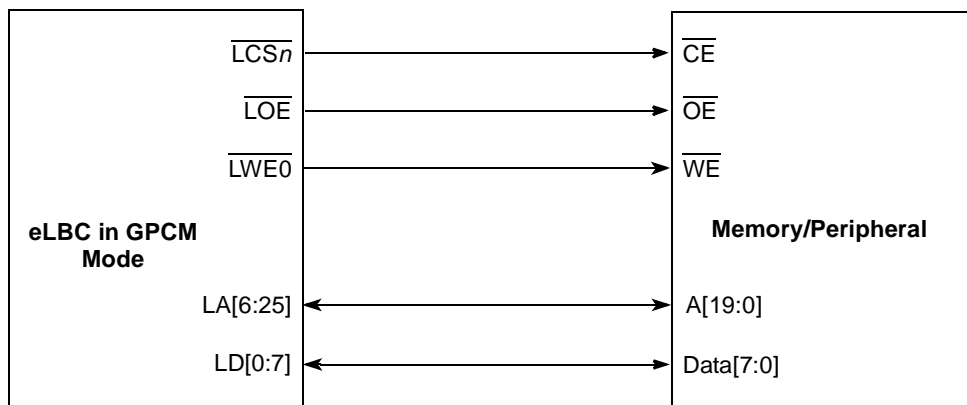


Figure 10-32. Enhanced Local Bus to GPCM Device Interface

Note that  $\overline{LWE0}$  is used for  $LD[0:7]$  and  $\overline{LWE1}$  is used for  $LD[8:15]$  for an 8-bit access to a 16-bit device.

Figure 10-33 shows  $\overline{LCS}$  as defined by the setup time required between the address lines and  $\overline{CE}$ . The user can configure  $OR_n[ACS]$  to specify  $\overline{LCS}$  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.

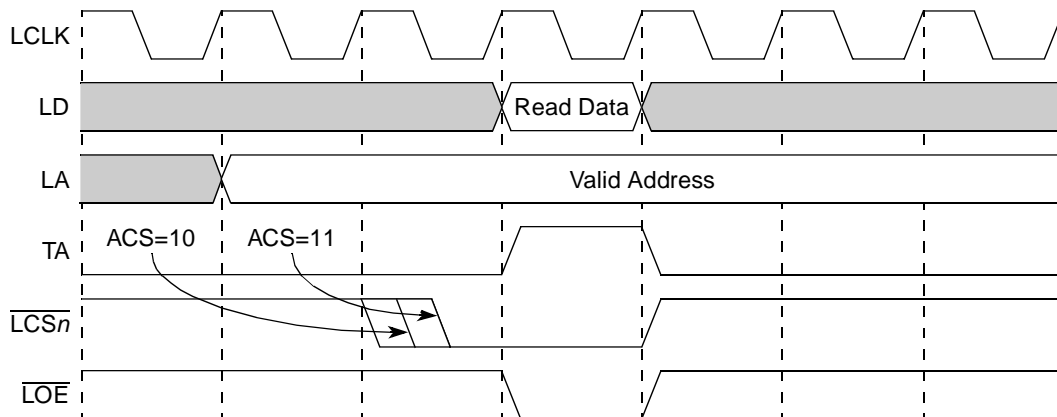
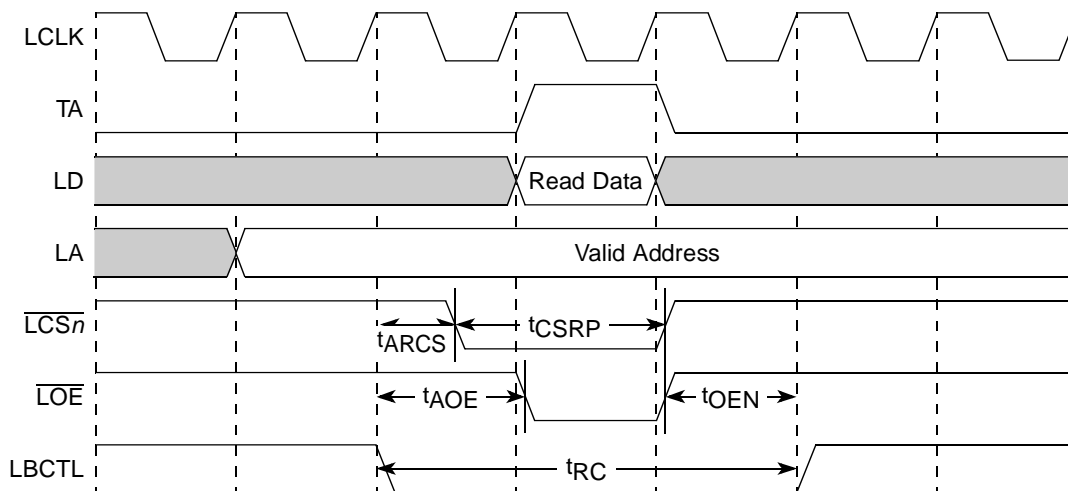


Figure 10-33. GPCM Basic Read Timing (XACS = 0, ACS = 1x, TRLX = 0, CLKDIV = 4,8)

### 10.4.2.1 GPCM Read Signal Timing

The basic GPCM read timing parameters that may be set by the  $OR_n$  attributes are shown in Figure 10-34. The read access cycle commences upon latching of the memory address 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.  $\overline{LOE}$  and  $\overline{LCSn}$  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.

$t_{CSRP}$  = Read chip-select assertion period.

$t_{ARCS}$  = Address valid to read chip-select time.  $t_{OEN}$  = Output enable negated time.

$t_{AOE}$  = Address valid to output enable time.

Figure 10-34. GPCM General Read Timing Parameters

Table 10-32 lists the signal timing parameters for a GPCM read access as the option register attributes are varied.

Table 10-32. GPCM Read Control Signal Timing

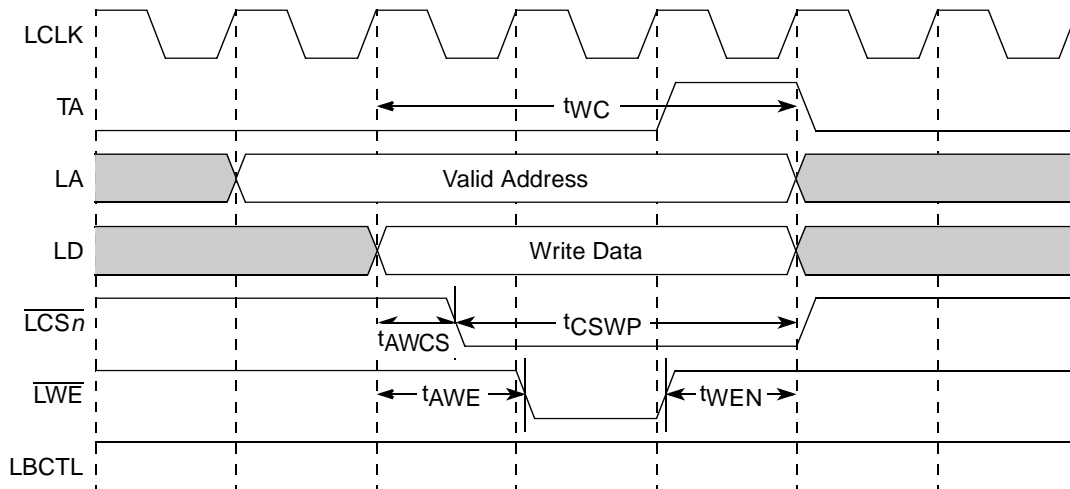
Option Register Attributes				Signal Timing (LCLK clock cycles) <sup>1</sup>				
TRLX	EHTR	XACS	ACS	t <sub>ARCS</sub>	t <sub>CSR<sub>P</sub></sub>	t <sub>AOE</sub>	t <sub>OEN</sub>	t <sub>RC</sub>
0	0	0	0X	0	2+SCY	1	0	2+SCY
0	0	0	10	¼ (½)	1¾+SCY (2+SCY)	1	0	2+SCY
0	0	0	11	½	1½+SCY	1	0	2+SCY
0	0	1	0X	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 (½)	1¾+SCY (1½+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+2xSCY	1	4	6+2xSCY
1	0	0	10	1¼ (1½)	1¾+2xSCY (1½+2xSCY)	2	4	7+2xSCY
1	0	0	11	1½	1½+2xSCY	2	4	7+2xSCY
1	0	1	0X	0	2+2xSCY	1	4	6+2xSCY
1	0	1	10	2	1+2xSCY	2	4	7+2xSCY
1	0	1	11	3	1+2xSCY	3	4	8+2xSCY
1	1	0	0X	0	2+2xSCY	1	8	10+2xSCY
1	1	0	10	1¼ (1½)	1¾+2xSCY (1½+2xSCY)	2	8	11+2xSCY
1	1	0	11	1½	1½+2xSCY	2	8	11+2xSCY
1	1	1	0X	0	2+2xSCY	1	8	10+2xSCY
1	1	1	10	2	1+2xSCY	2	8	11+2xSCY
1	1	1	11	3	1+2xSCY	3	8	12+2xSCY

<sup>1</sup> Times in parentheses are specific for the case LCRR[CLKDIV] = 2; other times apply to all CLKDIV values.



### 10.4.2.2 GPCM Write Signal Timing

The basic GPCM write timing parameters that may be set by the  $OR_n$  attributes are shown in Figure 10-35. The write access cycle commences upon latching of the memory address, and concludes when  $\overline{LCSn}$  returns high.  $\overline{LBCTL}$  remains stable for the entire cycle to drive data onto any secondary data bus. Write data becomes invalid following the falling edge of  $TA$ .  $\overline{LWE}$  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 write chip-select negation time.

Figure 10-35. GPCM General Write Timing Parameters

Table 10-33 lists the signal timing parameters for a GPCM write access as the option register attributes are varied.

Table 10-33. GPCM Write Control Signal Timing

Option Register Attributes				Signal Timing (LCLK clock cycles) <sup>1</sup>				
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	$\frac{1}{4}$ ( $\frac{1}{2}$ )	$1\frac{3}{4}$ +SCY (2+SCY)	1	0	2+SCY
0	0	11	0	$\frac{1}{2}$	$1\frac{1}{2}$ +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	$\frac{1}{4}$ (0)	2+SCY
0	0	10	1	$\frac{1}{4}$ ( $\frac{1}{2}$ )	$1\frac{1}{2}$ +SCY	1	0	$1\frac{3}{4}$ +SCY ( $1\frac{1}{2}$ +SCY)

Table 10-33. GPCM Write Control Signal Timing (continued)

Option Register Attributes				Signal Timing (LCLK clock cycles) <sup>1</sup>				
TRLX	XACS	ACS	CSNT	t <sub>AWCS</sub>	t <sub>CSWP</sub>	t <sub>AWE</sub>	t <sub>WEN</sub>	t <sub>WC</sub>
0	0	11	1	½	1¼+SCY (1+SCY)	1	0	1¾+SCY (1½+SCY)
0	1	00	1	0	2+SCY	1	¼ (0)	2+SCY
0	1	10	1	1	¾+SCY (½+SCY)	1	0	1¾+SCY (1½+SCY)
0	1	11	1	2	¾+SCY (½+SCY)	2	0	2¾+SCY (2½+SCY)
1	0	00	0	0	2+2xSCY	1	0	2+2xSCY
1	0	10	0	1¼ (1½)	1¾+2xSCY (2+2xSCY)	2	0	3+2xSCY
1	0	11	0	1½	1½+2xSCY	2	0	3+2xSCY
1	1	00	0	0	2+2xSCY	1	0	2+2xSCY
1	1	10	0	2	1+2xSCY	2	0	3+2xSCY
1	1	11	0	3	1+2xSCY	3	0	4+2xSCY
1	0	00	1	0	3+2xSCY	1	¼ (1)	3+2xSCY
1	0	10	1	1¼ (1½)	1½+2xSCY	2	0	2¾+2xSCY (2½+2xSCY)
1	0	11	1	1½	1¼+2xSCY (1+2xSCY)	2	0	2¾+2xSCY (2½+2xSCY)
1	1	00	1	0	3+2xSCY	1	¼ (1)	3+2xSCY
1	1	10	1	2	¾+2xSCY (½+2xSCY)	2	0	2¾+2xSCY (2½+2xSCY)
1	1	11	1	3	¾+2xSCY (½+2xSCY)	3	0	3¾+2xSCY (3½+2xSCY)

<sup>1</sup> Times in parentheses are specific for the case LCRR[CLKDIV] = 2; other times apply to all CLKDIV values.

### 10.4.2.3 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:

- One quarter of a clock cycle later (for LCRR[CLKDIV] = 4, 8).
- One half of a clock cycle later (for LCRR[CLKDIV] = 2, 4, or 8).
- One clock cycle later (for LCRR[CLKDIV] = 4), when OR<sub>n</sub>[XACS] = 1.
- Two clock cycles later (for LCRR[CLKDIV] = 2, 4, or 8), when OR<sub>n</sub>[XACS] = 1.

- Three clock cycles later (for LCRR[CLKDIV] = 2, 4, or 8), when  $ORn[XACS] = 1$  and  $ORn[TRLX] = 1$ .

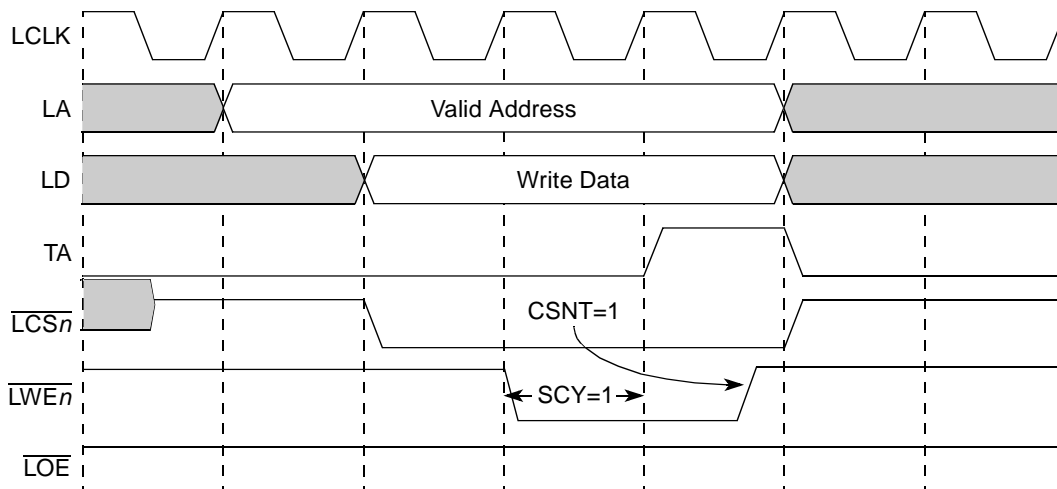
The timing diagram in [Figure 10-33](#) shows two chip-select assertion timings for the case LCRR[CLKDIV] = 4 or 8. If LCRR[CLKDIV] = 2,  $\overline{LCSn}$  asserts identically for  $ORn[ACS] = 10$  or 11.

#### 10.4.2.3.1 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  $ORn[SCY]$  and  $ORn[TRLX]$ . Internal generation of transfer acknowledge is enabled if  $ORn[SETA] = 0$ . If  $\overline{LGTA}$  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  $\overline{LGTA}$ ; otherwise it is terminated by the expiration of the wait state counter. Regardless of the setting of  $ORn[SETA]$ , wait states prolong the assertion duration of both  $\overline{LOE}$  and  $\overline{LWEn}$  in the same manner. When  $TRLX = 1$ , the number of wait states inserted by the memory controller is doubled from  $ORn[SCY]$  cycles to  $2 \times ORn[SCY]$  cycles, allowing a maximum of 30 wait states.

#### 10.4.2.3.2 Chip-Select and Write Enable Negation Timing

[Figure 10-32](#) shows a basic connection between the local bus and a static memory device. In this case,  $\overline{LCSn}$  is connected directly to  $\overline{CE}$  of the memory device. The  $\overline{LWE}[0:1]$  signals are connected to the respective  $\overline{WE}[1:0]$  signals on the memory device where each  $\overline{LWE}[0:1]$  signal corresponds to a different data byte.



**Figure 10-36. GPCM Basic Write Timing**  
(XACS = 0, ACS = 00, CSNT = 1, SCY = 1, TRLX = 0, CLKDIV = 2, 4, 8)

The strobes for the transaction are supplied by  $\overline{LOE}$  or  $\overline{LWEn}$ , depending on the transaction direction—read or write (write case shown in the figure).  $ORn[CSNT]$ , along with  $ORn[TRLX]$ , 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 provided that LCRR[CLKDIV] = 2, 4, or 8. For

example, when  $ACS = 00$  and  $CSNT = 1$ ,  $\overline{LWEn}$  is negated one quarter of a clock earlier, as shown in [Figure 10-36](#). If  $LCRR[CLDIV] = 2$ ,  $\overline{LWEn}$  is negated coincident with  $\overline{LCSn}$ .

1.  $\overline{LCSn}$  is affected by  $CSNT$  and  $TRLX$  only if  $ACS[0]$  is non zero. However,  $\overline{LWEn}$  is affected independent of  $ACS$ .
2. When  $CSNT$  attribute is asserted, the strobe is negated one quarter of a clock before the normal case provided that  $LCRR[CLDIV] = 4$  or  $8$ .
3.  $TRLX = 1$  in conjunction with  $CSNT = 1$ , negates the  $\overline{LWEn}$   $1+1/4$  cycle earlier if  $LCRR[CLKDIV] = 4$  or  $8$ .

If  $LCRR[CLKDIV] = 2$ ,  $\overline{LCSn}$  and  $\overline{LWEn}$  are negated one cycle earlier if  $TRLX = 1$ .

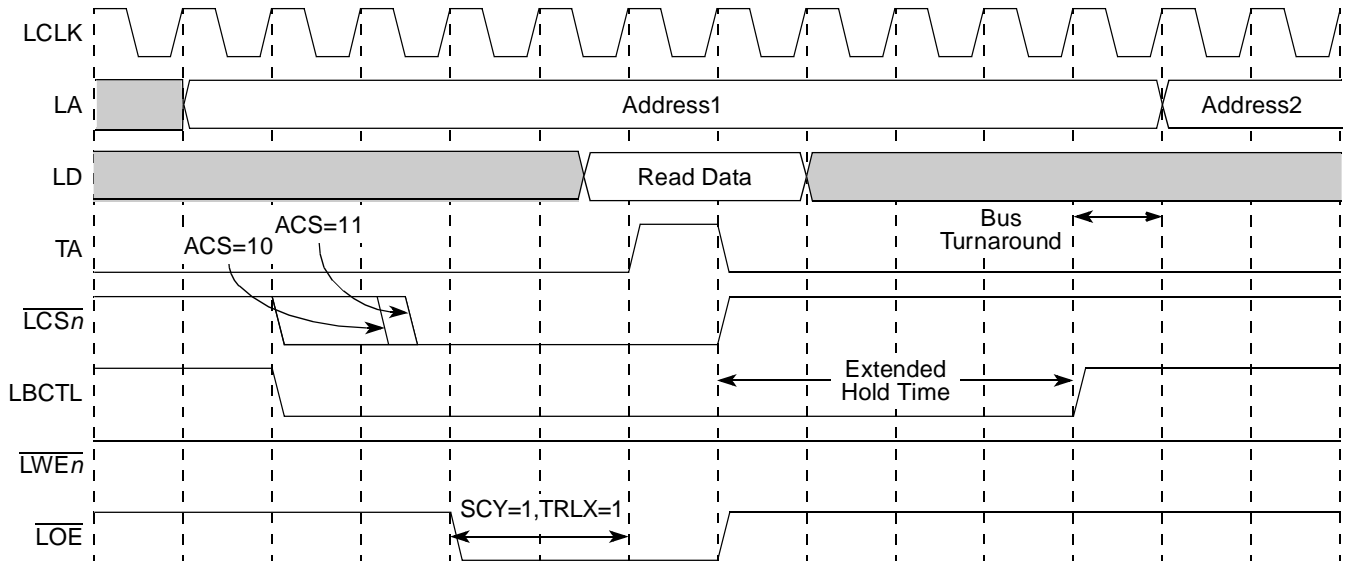
For example, when  $ACS = 00$ ,  $CSNT = 1$  and  $TRLX = 0$ ,  $\overline{LWEn}$  is negated one quarter of a clock earlier and  $\overline{LCSn}$  is negated normally as shown in [Figure 10-36](#).

#### 10.4.2.3.3 Relaxed Timing

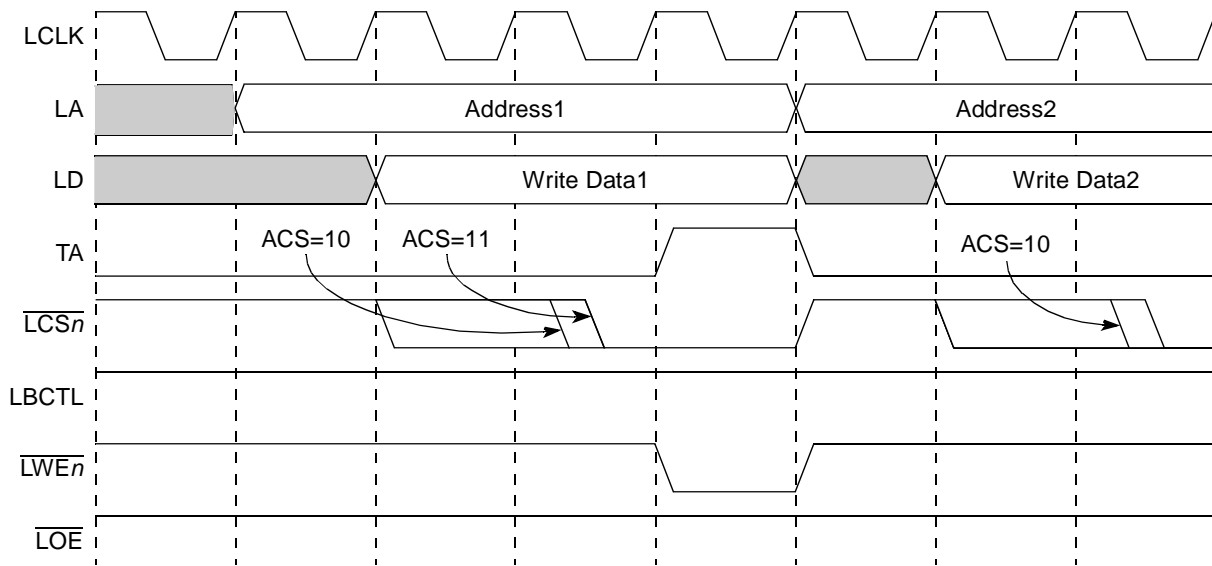
$ORx[TRLX]$  is provided for memory systems that require more relaxed timing between signals. Setting  $TRLX = 1$  has the following effect on timing:

- 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.
- $\overline{LCSn}$  signals are negated one cycle earlier during writes (but only if  $ACS$  is not equal to  $00$ ).
- $\overline{LWE}[0:1]$  signals are negated one cycle earlier during writes.

[Figure 10-37](#) and [Figure 10-38](#) show relaxed timing read and write transactions. The effect of  $LCRR[CLKDIV] = 2$  for these examples is only to delay the assertion of  $\overline{LCSn}$  in the  $ACS = 10$  case to the  $ACS = 11$  case. The example in [Figure 10-38](#) also shows address and data multiplexing on  $LD$  for a pair of writes issued consecutively.

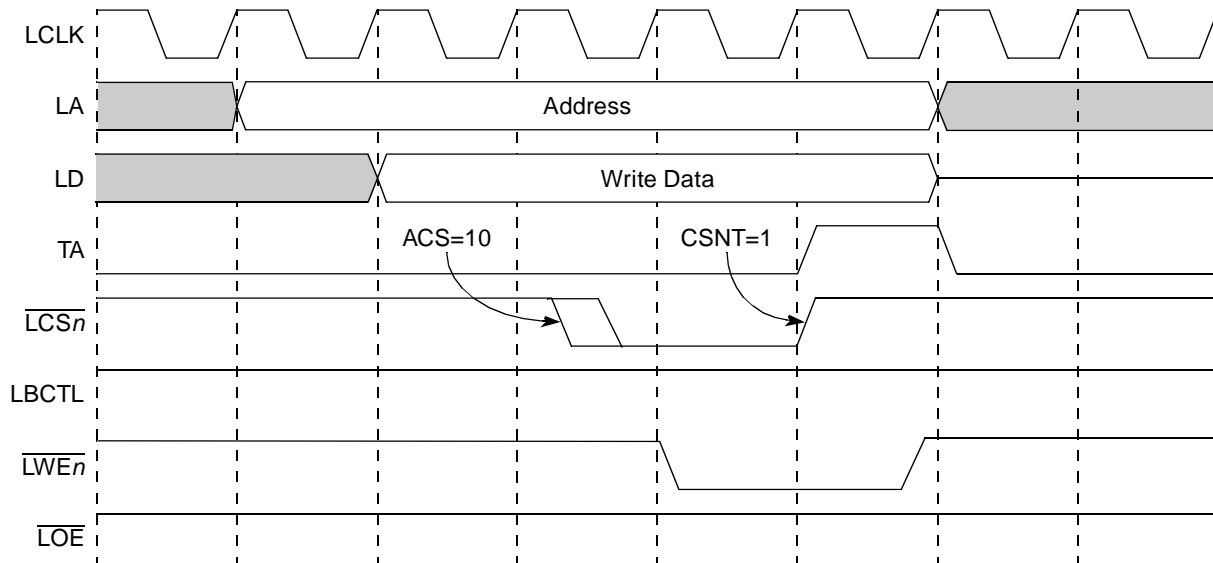


**Figure 10-37. GPCM Relaxed Timing Back-to-Back Reads**  
 (XACS = 0, ACS = 1x, SCY = 1, CSNT = 0, TRLX = 1, EHTR = 0, CLKDIV = 4, 8)

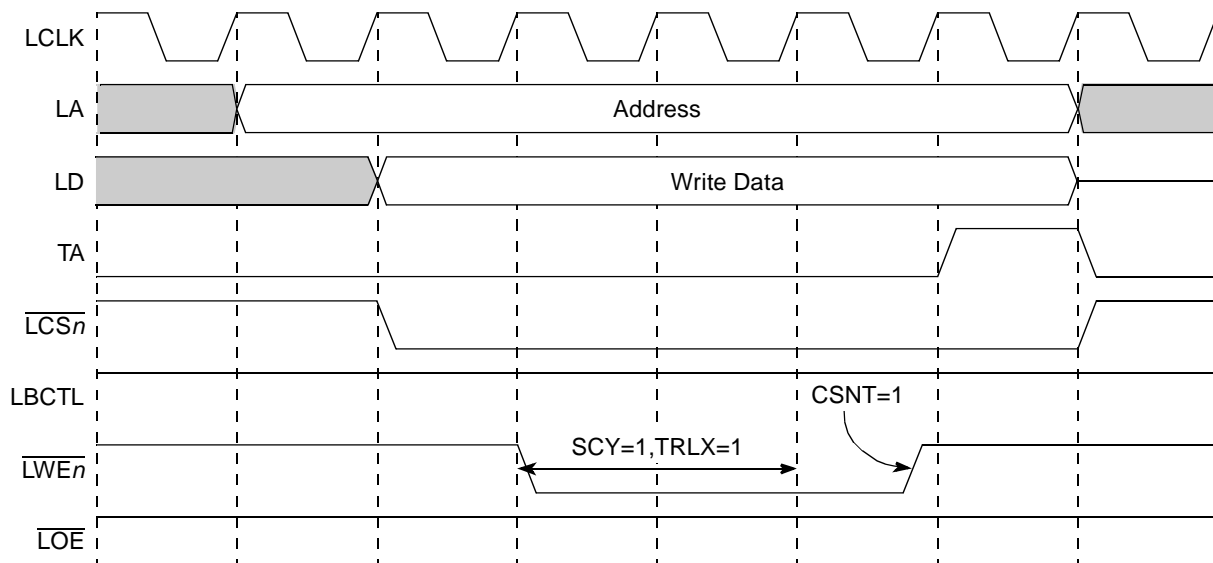


**Figure 10-38. GPCM Relaxed Timing Back-to-Back Writes**  
 (XACS = 0, ACS = 1x, SCY = 0, CSNT = 0, TRLX = 1, CLKDIV = 4, 8)

When  $\overline{\text{TRLX}}$  and  $\text{CSNT}$  are set in a write access, the  $\overline{\text{LWE}}[0:1]$  strobe signals are negated one clock earlier than in the normal case, as shown in Figure 10-39 and Figure 10-40. If  $\text{ACS} \neq 00$ ,  $\overline{\text{LCSn}}$  is also negated one clock earlier.



**Figure 10-39. GPCM Relaxed Timing Write**  
( $\text{XACS} = 0$ ,  $\text{ACS} = 10$ ,  $\text{SCY} = 0$ ,  $\text{CSNT} = 1$ ,  $\overline{\text{TRLX}} = 1$ ,  $\text{CLKDIV} = 4, 8$ )



**Figure 10-40. GPCM Relaxed Timing Write**  
( $\text{XACS} = 0$ ,  $\text{ACS} = 00$ ,  $\text{SCY} = 1$ ,  $\text{CSNT} = 1$ ,  $\overline{\text{TRLX}} = 1$ ,  $\text{CLKDIV} = 4, 8$ )

#### 10.4.2.3.4 Output Enable ( $\overline{\text{LOE}}$ ) Timing

The timing of the  $\overline{\text{LOE}}$  is affected only by  $\overline{\text{TRLX}}$ . It always asserts and negates on the rising edge of the bus clock.  $\overline{\text{LOE}}$  asserts either on the rising edge of the bus clock after  $\overline{\text{LCSn}}$  is asserted or coinciding with  $\overline{\text{LCSn}}$  (if  $\text{XACS} = 1$  and  $\text{ACS} = 10$  or  $\text{ACS} = 11$ ). Accordingly, assertion of  $\overline{\text{LOE}}$  can be delayed (along

with the assertion of  $\overline{\text{LCSn}}$  by programming  $\text{TRLX} = 1$ .  $\overline{\text{LOE}}$  negates on the rising clock edge coinciding with  $\overline{\text{LCSn}}$  negation

#### 10.4.2.3.5 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  $\text{ORn}[\text{TRLX}, \text{EHTR}]$ . Any access following a read access to the slower memory bank is delayed by the number of clock cycles specified in [Table 10-7](#) 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  $\text{ORn}[\text{EHTR}]$ .

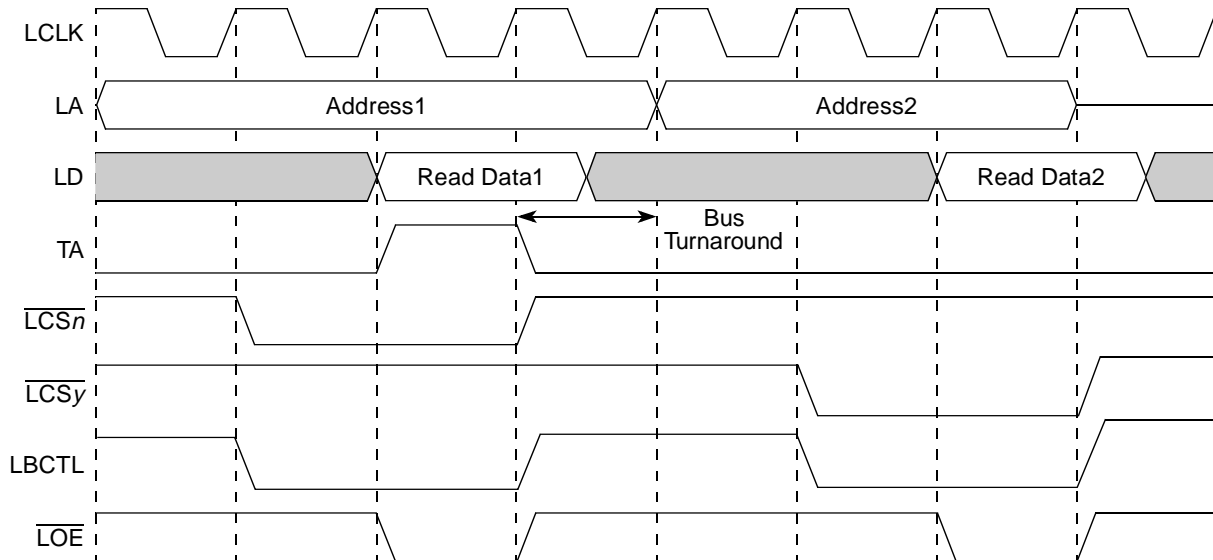


Figure 10-41. GPCM Read Followed by Read ( $\text{TRLX} = 0$ ,  $\text{EHTR} = 0$ , Fastest Timing)

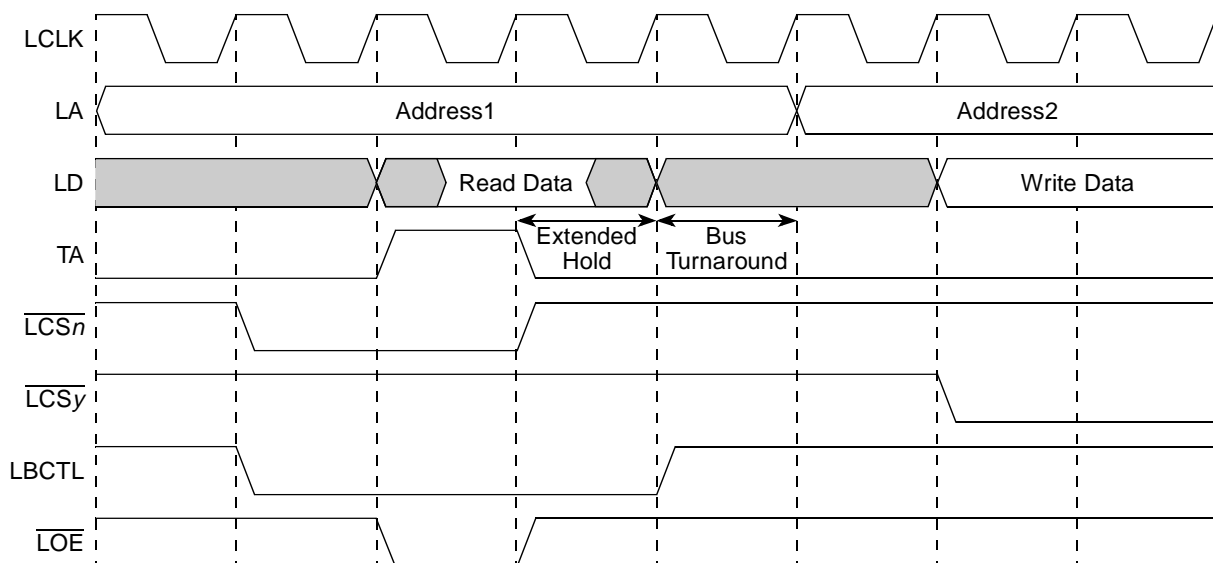


Figure 10-42. GPCM Read Followed by Write  
( $\text{TRLX} = 0$ ,  $\text{EHTR} = 1$ , One-Cycle Extended Hold Time on Reads)

### 10.4.2.4 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  $\overline{\text{LCSn}}$ , 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  $\text{ORn}[\text{SETA}]$ ).  $\overline{\text{LGTA}}$  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  $\text{ORn}[\text{SETA}]$ . Asserting  $\overline{\text{LGTA}}$  always terminates an access, even if  $\text{ORn}[\text{SETA}] = 0$  (internal transfer acknowledge generation), but it is the only means by which an access can be terminated if  $\text{ORn}[\text{SETA}] = 1$ . The timing of  $\overline{\text{LGTA}}$  is illustrated by the example in Figure 10-43.

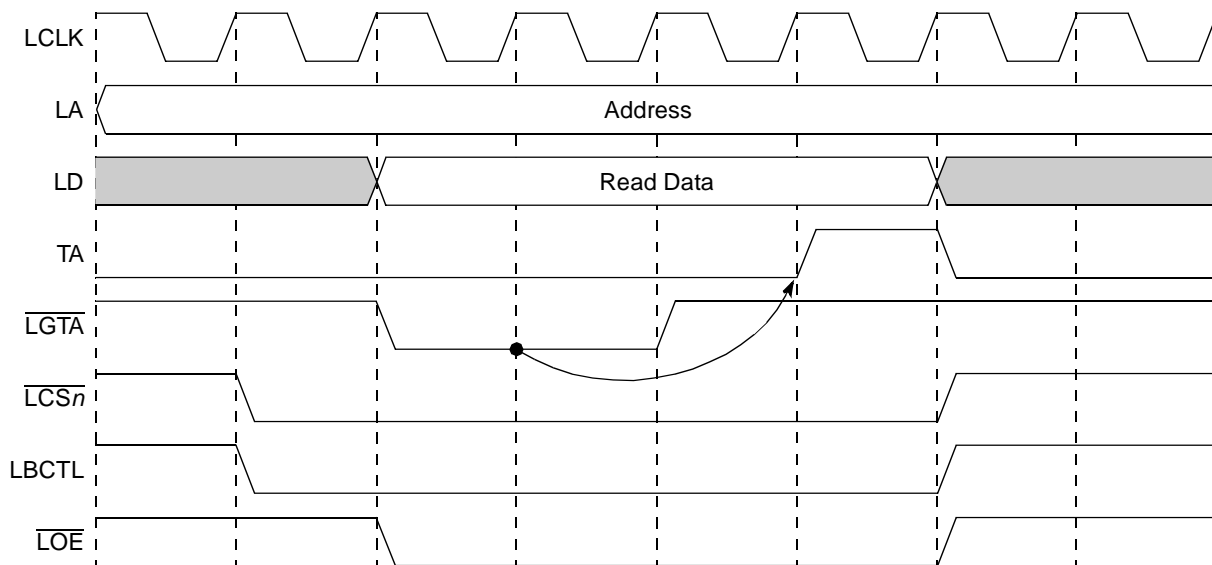


Figure 10-43. External Termination of GPCM Access

### 10.4.2.5 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  $\text{BR0}$  or  $\text{OR0}$  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  $\text{OR0}$  and it can be used as any other chip-select register after the preferred address range is loaded into  $\text{BR0}$ . After the first write to  $\text{OR0}$ , the boot chip-select can be restarted only with a hardware reset. Table 10-34 describes the initial values of the boot bank in the memory controller.



**Table 10-34. Boot Bank Field Values after Reset for GPCM as Boot Controller**

Register	Field	Setting
BR0	BA	0000_0000_0000_0000_0
	PS	<i>From RCWH[ROMLOC], RLEXT = 0</i>
	DECC	00
	WP	0
	MSEL	000
	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

### 10.4.3 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<sub>n</sub>, OR<sub>n</sub>, and FMR.

Figure 10-44 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 LD[0:7]<sup>1</sup>, with  $\overline{\text{LFWEO}}$  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-K $\Omega$  resistor. On system reset, a global (boot) chip-select is available that provides a boot ROM chip-select (LCS0) prior to the system being fully configured.

1. Note bit numbering reversal: LD[0] (msb) connects to Flash IO[7], while LD[7] (lsb) connects to IO[0].

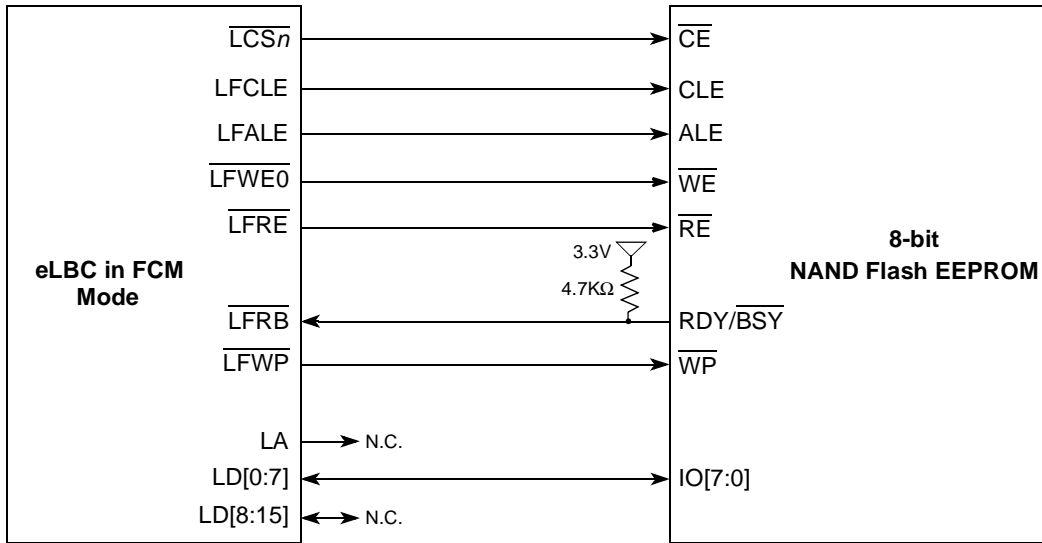


Figure 10-44. Local Bus to 8-Bit FCM Device Interface

Basic read access timing for FCM is shown in Figure 10-45. Although LCLK is shown for reference, NAND Flash EEPROMs do not make use of the clock.

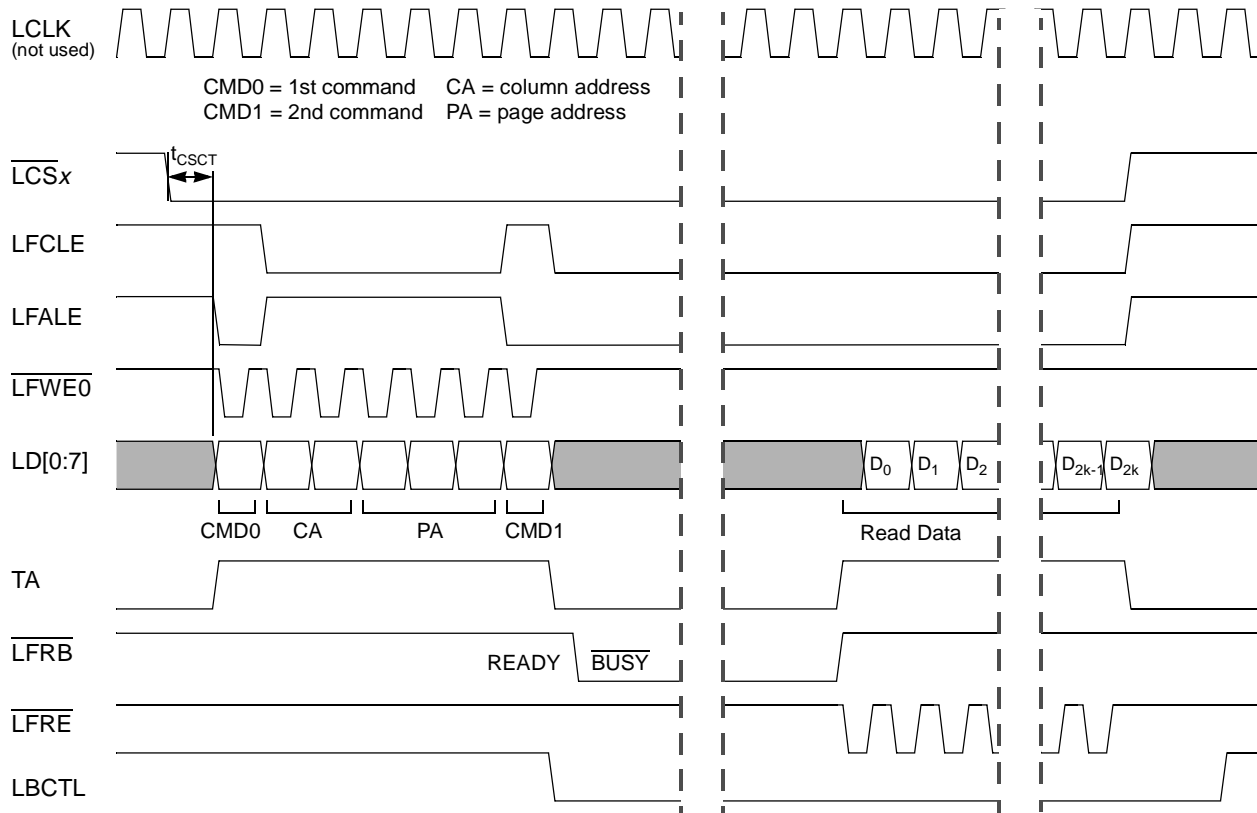


Figure 10-45. FCM Basic Page Read Timing  
(PGS = 1, CSCT = 0, CST = 0, CHT = 1, RST = 1, SCY = 0, TR LX = 0, EHTR = 1)

FCM asserts  $\overline{LCSn}$  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  $\overline{LFWEO}$ , 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  $\overline{LFRB}$  low to indicate that the device is busy. Since in [Figure 10-45](#) FCM is now expecting a read response, it takes  $\overline{LBCTL}$  low to turnaround any bus transceivers that are present. Upon  $\overline{LFRB}$  indicating ready status, FCM asserts  $\overline{LFRE}$  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  $\overline{LCSn}$  and delays eLBC by  $t_{EHTR}$  before any subsequent memory access occurs.

### 10.4.3.1 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 signal, such as  $\overline{LCSn}$ , 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  $ORn[PGS]$ , 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 executes 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 executes 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  $\overline{LFWP}$  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 10.4.3.4.2, “Boot Block Loading into the FCM Buffer RAM,”](#) for a description of the shared buffer RAM layout during boot.

#### 10.4.3.1.1 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 ( $ORn[PGS] = 0$ ), mapped as shown in [Figure 10-46](#). 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  $(P \bmod 8)$ , where  $P = \text{FPAR}[\text{PI}]$ . Since the bank size set by  $\text{OR}_n[\text{AM}]$  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  $\text{FBCR}[\text{BC}] = 0$ , 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  $\text{FBCR}[\text{BC}]$ ,  $\text{FPAR}[\text{MS}]$  locates the starting address in either the main region ( $\text{MS} = 0$ ) or the spare region ( $\text{MS} = 1$ ). 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.

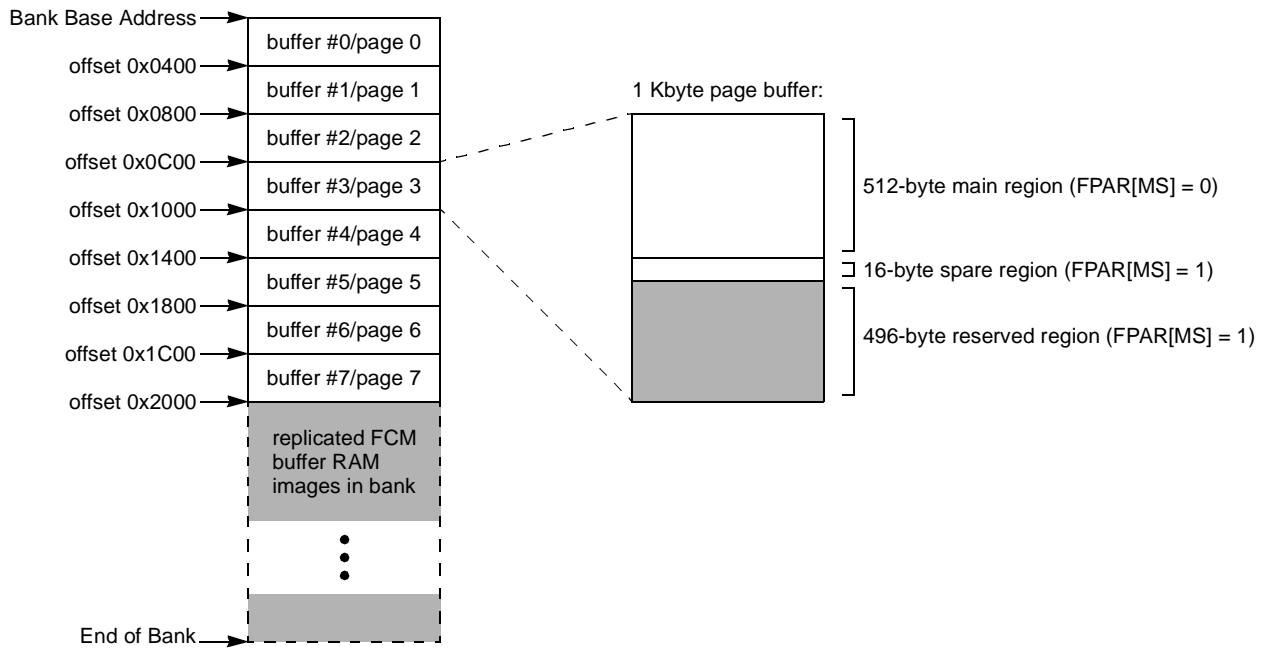


Figure 10-46. FCM Buffer RAM Memory Map for Small-Page (512-byte page) NAND Flash Devices

#### 10.4.3.1.2 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 ( $\text{OR}_n[\text{PGS}] = 1$ ), mapped as shown in Figure 10-47. 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  $(P \bmod 2)$ , where  $P = \text{FPAR}[\text{PI}]$ . Since the bank size set by  $\text{OR}_n[\text{AM}]$  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  $\text{FBCR}[\text{BC}] = 0$ , 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  $\text{FBCR}[\text{BC}]$ ,  $\text{FPAR}[\text{MS}]$  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.

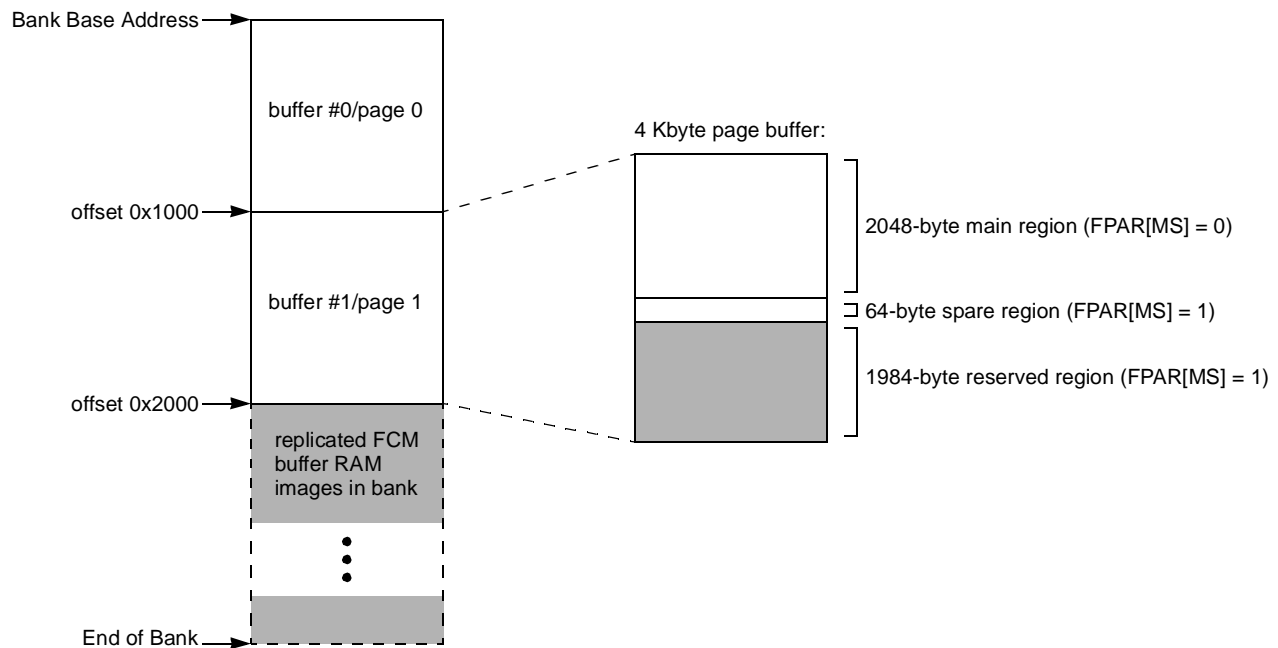


Figure 10-47. FCM Buffer RAM Memory Map for Large-Page (2-Kbyte page) NAND Flash Devices

### 10.4.3.1.3 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 10-48. 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).

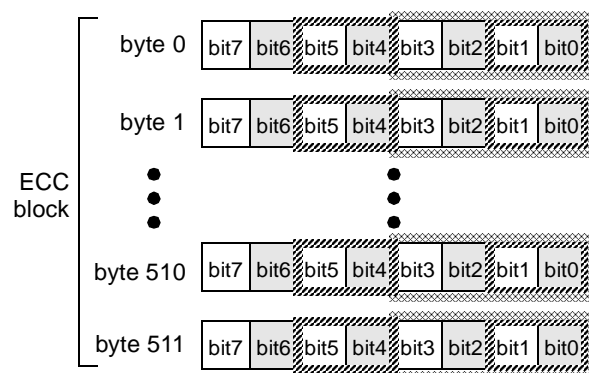


Figure 10-48. FCM ECC Calculation

The placement of ECC code words in relation to FMR[ECCM] is shown in Figure 10-49. 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 ( $BRn[DECC] = 10$ ), 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 10-49 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  $BRn[DECC] = 01$  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 10-49. ECC Placement in NAND Flash Spare Regions in Relation to FMR[ECCM]

### 10.4.3.2 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 10-50. 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  $\overline{LCSn}$ , otherwise,  $\overline{LCSn}$  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 10.3.1.18, "Flash Instruction Register (FIR)."

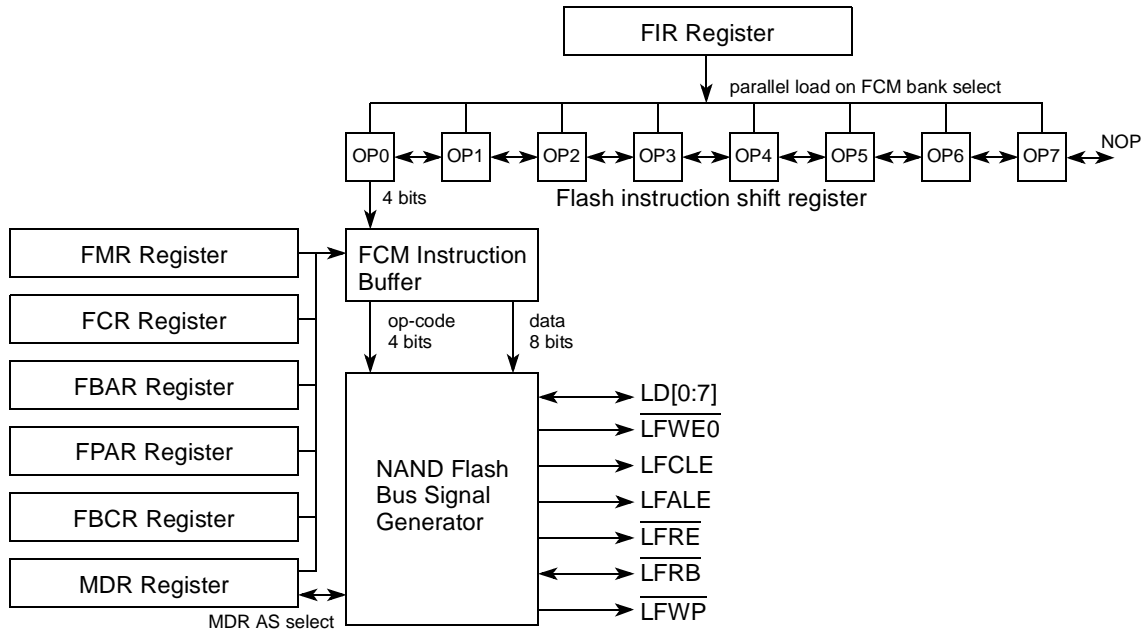


Figure 10-50. FCM Instruction Sequencer Mechanism

#### 10.4.3.2.1 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  $\overline{\text{LFWEO}}$  while driving an 8-bit command onto LD[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  $\overline{\text{LFRB}}$  to be sampled high (EEPROM in ready state) before issuing—CW0, and CW1. These commands first poll the  $\overline{\text{LFRB}}$  pin, waiting for it to go high, before writing a single command byte onto LD[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  $\overline{\text{LFRB}}$ . Rather, FCM waits for  $8 \times (2 + \text{OR}_n[\text{SCY}])$  clock cycles (when  $\text{OR}_n[\text{TRLX}] = 0$ ) or  $16 \times (2 + \text{OR}_n[\text{SCY}])$  clock cycles (when  $\text{OR}_n[\text{TRLX}] = 1$ ) before sampling the level of  $\overline{\text{LFRB}}$ . If the level of  $\overline{\text{LFRB}}$  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.

#### 10.4.3.2.2 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  $\overline{\text{LFWEO}}$  are asserted. Thus a NOP instruction may be used to insert a pause matching the time taken for a regular command write.

### 10.4.3.2.3 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 LD[0:7] with LFALE and  $\overline{\text{LFWE0}}$  asserted together. There are three kinds of address generation provided:

- Column address—CA. A column address comprises one byte ( $\text{OR}_n[\text{PGS}] = 0$ ) or two bytes ( $\text{OR}_n[\text{PGS}] = 1$ ) 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.

### 10.4.3.2.4 FCM Data Read Instructions

Data read instructions assert  $\overline{\text{LFRE}}$  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  $\overline{\text{LFRE}}$  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  $\overline{\text{LFRB}}$  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  $\overline{\text{LFRB}}$  pin follow the behavior of CW<sub>n</sub> instructions.
- Read data/status to MDR once waited on ready—RSW. This instruction first polls the  $\overline{\text{LFRB}}$  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  $\overline{\text{LFRB}}$  pin follow the behavior of  $\text{CW}_n$  instructions.

#### 10.4.3.2.5 FCM Data Write Instructions

Data write instructions assert  $\overline{\text{LFWEO}}$  repeatedly (with  $\text{LFCLE}$  and  $\text{LFALE}$  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  $\text{FBCR}[\text{BC}]$  bytes of data from the current FCM RAM buffer addressed by  $\text{FPAR}$ . If  $\text{FBCR}[\text{BC}] = 0$ , 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  $\text{FECC}_n$  registers and spare region in accordance with the setting of  $\text{FMR}[\text{ECCM}]$ . If the value of  $\text{FBCR}[\text{BC}]$  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  $\overline{\text{LFWEO}}$  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.

#### 10.4.3.3 FCM Signal Timing

If  $\text{BR}_n[\text{MSEL}]$  selects the FCM, the attributes for the memory cycle are taken from  $\text{OR}_n$ . These attributes include the  $\text{CSCT}$ ,  $\text{CST}$ ,  $\text{CHT}$ ,  $\text{RST}$ ,  $\text{SCY}$ ,  $\text{TRLX}$ , and  $\text{EHTR}$  fields.

##### 10.4.3.3.1 FCM Chip-Select Timing

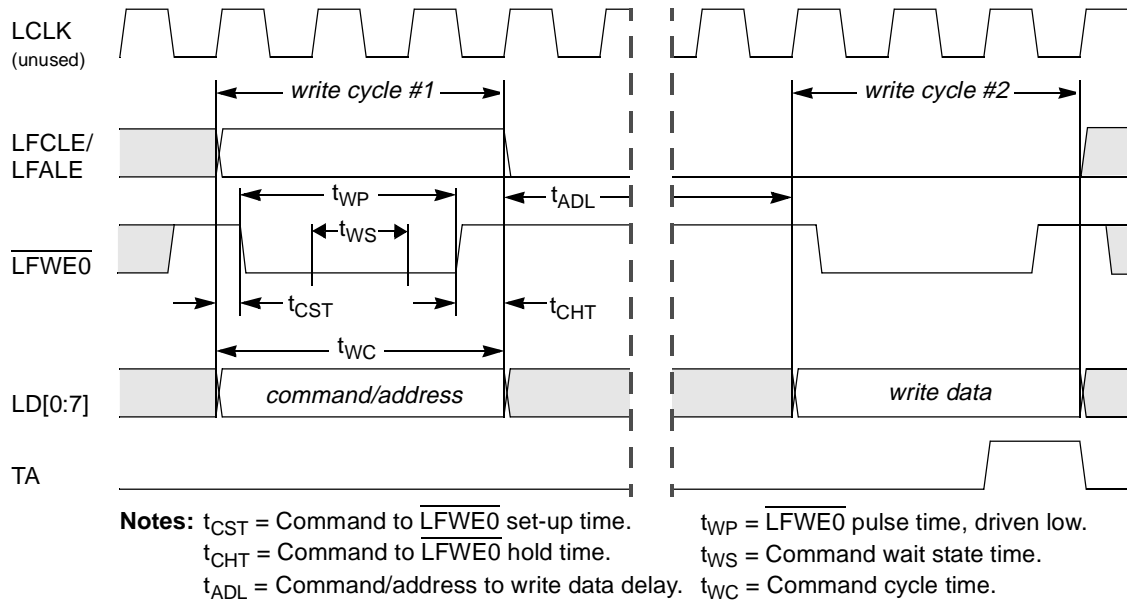
The timing of  $\overline{\text{LCS}}_n$  assertion in FCM mode is illustrated by the timing diagram in [Figure 10-45](#).  $\overline{\text{LCS}}_n$  remains asserted until the last instruction in FIR has completed. The delay,  $t_{\text{CSCT}}$ , between  $\overline{\text{LCS}}_n$  assertion and commencement of the first NAND Flash instruction is controlled by  $\text{OR}_n[\text{CSCT}]$  and  $\text{OR}_n[\text{TRLX}]$ , as shown in [Table 10-35](#).  $\text{OR}_n[\text{CSCT}]$  should be set in accordance with the NAND Flash EEPROM chip-select to  $\overline{\text{WE}}$  set-up time specification.

**Table 10-35. FCM Chip-Select to First Command Timing**

$\text{OR}_n[\text{TRLX}]$	$\text{OR}_n[\text{CSCT}]$	$\overline{\text{LCS}}_n$ 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

##### 10.4.3.3.2 FCM Command, Address, and Write Data Timing

The FCM command ( $\text{CM}_0$ – $\text{CM}_3$ ,  $\text{CW}_0$ ,  $\text{CW}_1$ ), address ( $\text{CA}$ ,  $\text{PA}$ ,  $\text{UA}$ ), and data write ( $\text{WB}$ ,  $\text{WS}$ ) instructions all share the same basic timing attributes. Assertion of  $\overline{\text{LFWEO}}$  initiates transfer via  $\text{LD}[0:7]$ , and the options in  $\text{OR}_n$  for FCM mode establish the set-up, hold, and wait state timings with respect to  $\overline{\text{LFWEO}}$ , as shown in [Figure 10-51](#).



**Figure 10-51. Timing of FCM Command/Address and Write Data Cycles**  
(for  $TRLX = 0$ ,  $CHT = 0$ ,  $CST = 1$ ,  $SCY = 1$ ,  $CLKDIV = 4*N$ )

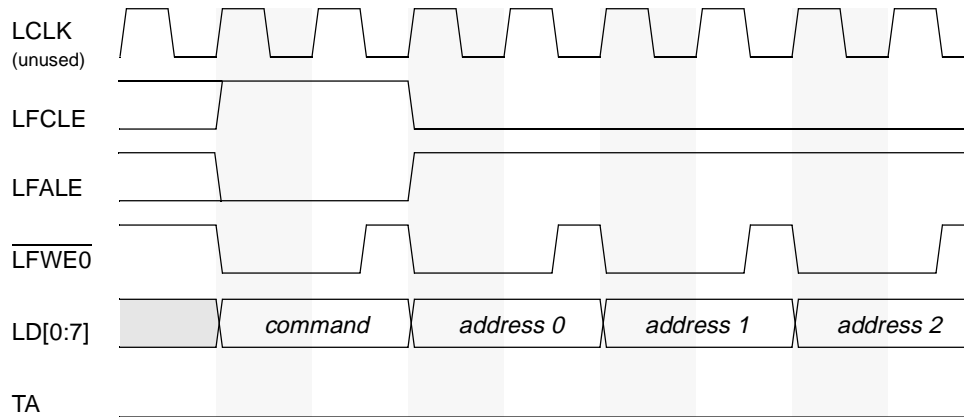
The timing parameters are summarized in [Table 10-36](#).

**Table 10-36. FCM Command, Address, and Write Data Timing Parameters**

Option Register Attributes			Timing Parameter (LCLK Clock Cycles) <sup>1</sup>					
TRLX	CHT	CST	$t_{CST}$	$t_{CHT}$	$t_{WS}$	$t_{WP}$	$t_{WC}$	$t_{ADL}$
0	0	0	0	$\frac{1}{2}$	SCY	$1\frac{1}{2}+SCY$	$2+SCY$	$4\times(2+SCY)$
0	0	1	$\frac{1}{4}$	$\frac{1}{2}$	SCY	$1\frac{1}{4}+SCY$	$2+SCY$	$4\times(2+SCY)$
0	1	0	0	1	SCY	$1+SCY$	$2+SCY$	$4\times(2+SCY)$
0	1	1	$\frac{1}{4}$	1	SCY	$\frac{3}{4}+SCY$	$2+SCY$	$4\times(2+SCY)$
1	0	0	$\frac{1}{2}$	$1\frac{1}{2}$	$2\times SCY$	$1+2\times SCY$	$3+2\times SCY$	$8\times(2+SCY)$
1	0	1	1	$1\frac{1}{2}$	$2\times SCY$	$\frac{1}{2}+2\times SCY$	$3+2\times SCY$	$8\times(2+SCY)$
1	1	0	$\frac{1}{2}$	2	$2\times SCY$	$\frac{1}{2}+2\times SCY$	$3+2\times SCY$	$8\times(2+SCY)$
1	1	1	1	2	$2\times SCY$	$2\times SCY$	$3+2\times SCY$	$8\times(2+SCY)$

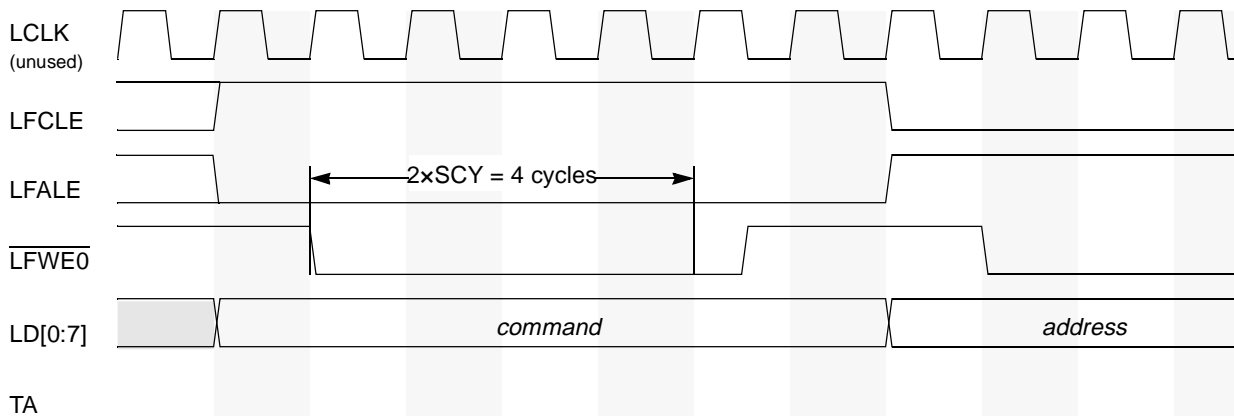
<sup>1</sup> In the parameters, SCY refers to a delay of  $ORn[SCY]$  clock cycles.

An example of minimum delay command timing appears in [Figure 10-52](#). Note that the set-up, wait-state, and hold timing of command, address, and write data cycles with respect to  $\overline{LFWE0}$  assertion are all identical, and that the minimum cycle extends for two LCLK clock cycles.



**Figure 10-52. Example of FCM Command and Address Timing with Minimum Delay Parameters (for  $TRLX = 0$ ,  $CHT = 0$ ,  $CST = 0$ ,  $SCY = 0$ ,  $CLKDIV = 4*N$ )**

An example of relaxed command timing is shown in [Figure 10-53](#).



**Figure 10-53. Example of FCM Command and Address Timing with Relaxed Parameters (for  $TRLX = 1$ ,  $CHT = 0$ ,  $CST = 1$ ,  $SCY = 2$ ,  $CLKDIV = 4*N$ )**

#### 10.4.3.3.3 FCM Ready/Busy Timing

Instructions CW0, CW1, RBW, and RSW force FCM to observe the state of the  $\overline{LFRB}$  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 10-54](#) before sampling the state of  $\overline{LFRB}$ . This guards against observing  $\overline{LFRB}$  before it has been properly driven low by the device, but does not preclude  $\overline{LFRB}$  from remaining high after a command. In addition, FCM samples and compares the state of  $\overline{LFRB}$  on two consecutive cycles of LCLK to filter out noise on this signal as it rises to the ready state ( $\overline{LFRB} = 1$ ).

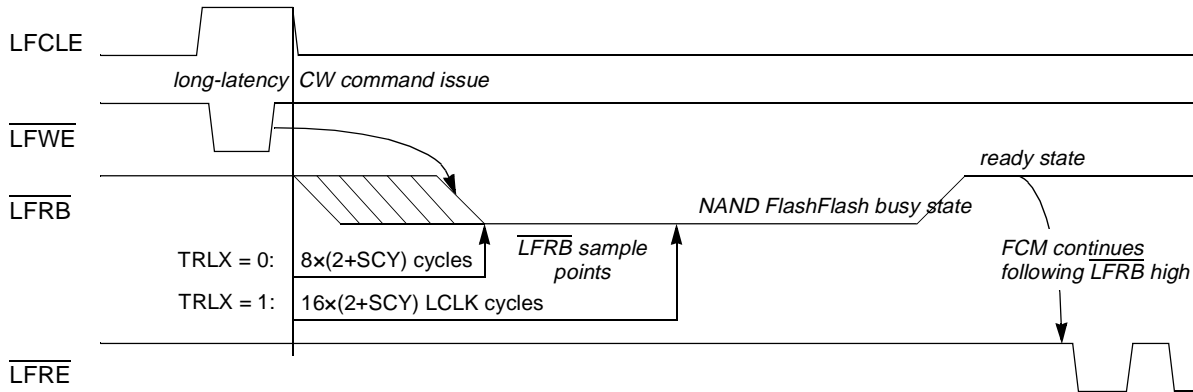
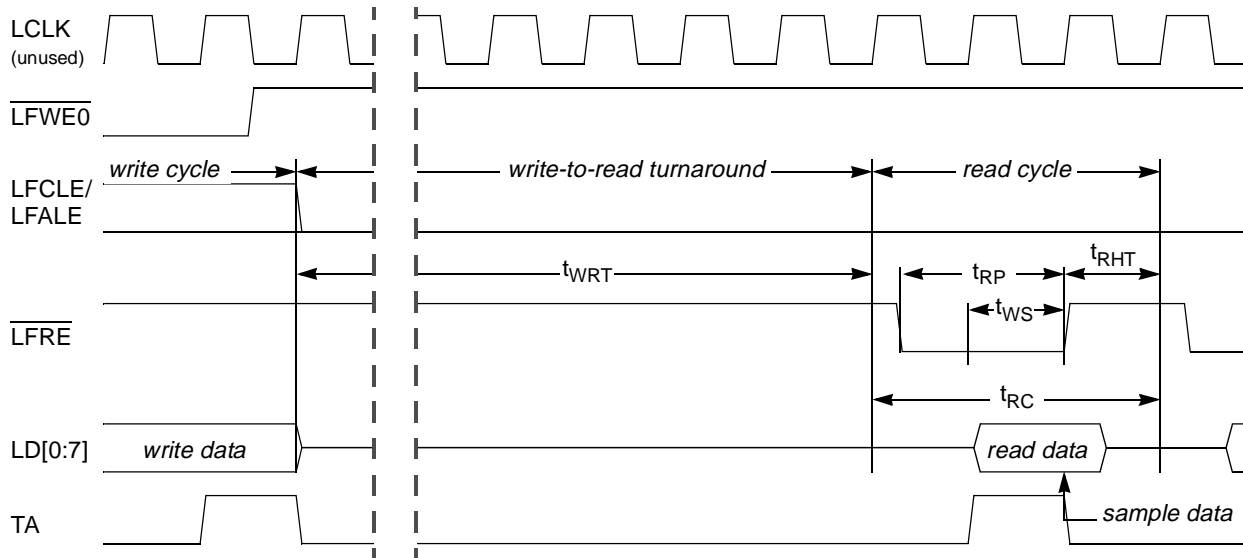


Figure 10-54. FCM Delay Prior to Sampling  $\overline{\text{LFRB}}$  State

### 10.4.3.3.4 FCM Read Data Timing

The timing for read data transfers is shown in Figure 10-55. 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  $\overline{\text{LFRB}}$  sample timing takes effect (see Section 10.4.3.3.3, “FCM Ready/Busy Timing”).



- Notes:**  $t_{RP}$  =  $\overline{\text{LFRE}}$  pulse time, read period.  $t_{WS}$  = Read wait state time.  
 $t_{RHT}$  =  $\overline{\text{LFRE}}$  hold time.  $t_{RC}$  = Read data cycle time.  
 $t_{WRT}$  = Write to read turnaround time.

Figure 10-55. FCM Read Data Timing  
 (for  $\text{TRLX} = 0$ ,  $\text{RST} = 0$ ,  $\text{SCY} = 1$ ,  $\text{CLKDIV} = 4*N$ )

The timing parameters are summarized in [Table 10-37](#).

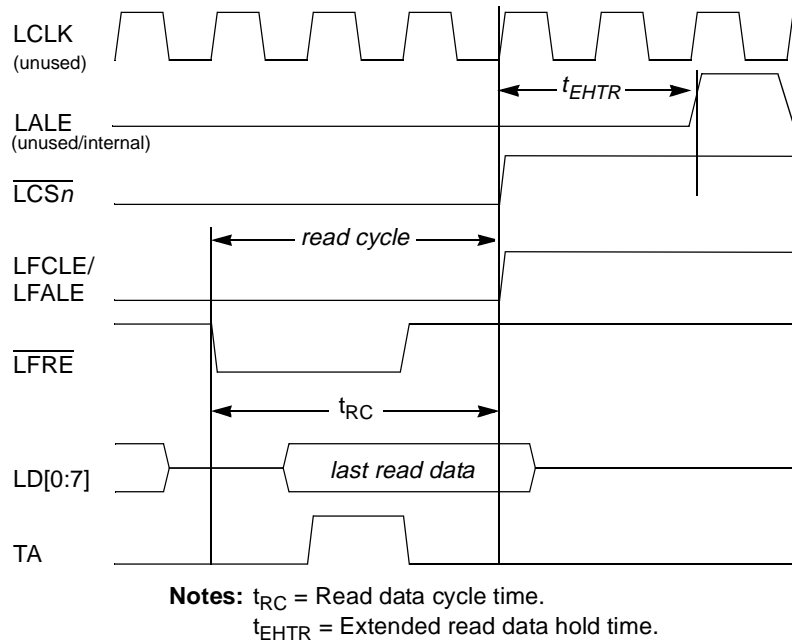
**Table 10-37. FCM Read Data Timing Parameters**

Option Register Attributes		Timing Parameter (LCLK Clock Cycles) <sup>1</sup>				
TRLX	RST	$t_{RP}$	$t_{RHT}$	$t_{WS}$	$t_{RC}$	$t_{WRT}$
0	0	$\frac{3}{4}+SCY$	1	SCY	$2+SCY$	$4 \times (2+SCY)$
0	1	$1+SCY$	1	SCY	$2+SCY$	$4 \times (2+SCY)$
1	0	$\frac{1}{2}+2 \times SCY$	2	$2 \times SCY$	$3+2 \times SCY$	$8 \times (2+SCY)$
1	1	$1+2 \times SCY$	2	$2 \times SCY$	$3+2 \times SCY$	$8 \times (2+SCY)$

<sup>1</sup> In the parameters, SCY refers to a delay of  $OR_n[SCY]$  clock cycles.

### 10.4.3.3.5 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 10-45](#) and [Figure 10-56](#), is a delay inserted by FCM between the last data read and another eLBC memory access.  $\overline{LCSn}$  is negated during  $t_{EHTR}$  to allow external devices and bus transceivers time to disable their drivers.



**Figure 10-56. FCM Read Data Timing with Extended Hold Time**  
 (for  $TRLX = 0$ ,  $EHTR = 1$ ,  $RST = 1$ ,  $SCY = 1$ ,  $CLKDIV = 4 \times N$ )

### 10.4.3.4 FCM Boot Chip-Select Operation

Boot chip-select operation allows address decoding for a boot ROM before system initialization.  $\overline{LCS0}$  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.

#### 10.4.3.4.1 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 10-38 describes the initial values of the boot bank in the memory controller.

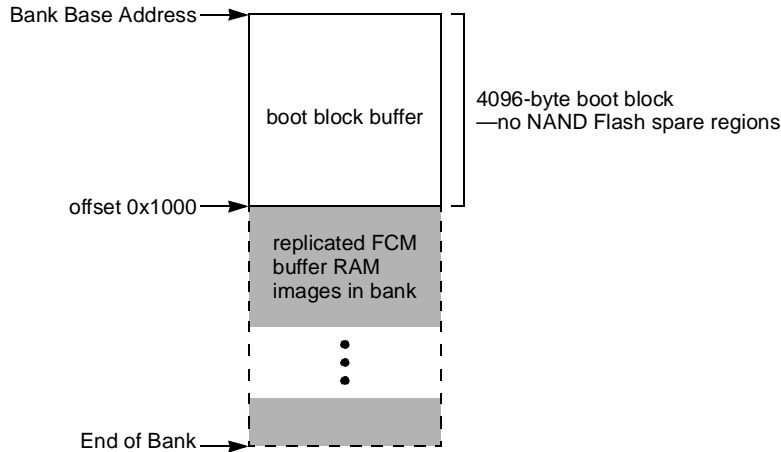
**Table 10-38. Boot Bank Field Values after Reset for FCM as Boot Controller**

Register	Field	Setting
BR0	BA	0000_0000_0000_0000_0
	PS	01
	DECC	00
	WP	0
	MSEL	001
	V	0
OR0	AM	0000_0000_0000_0000_0
	BCTLD	0
	PGS	From RCWH[ROMLOC], RLEXT = 01
	CSCT	1
	CST	1
	CHT	1
	RST	1
	SCY	010
	TRLX	1
	EHTR	

#### 10.4.3.4.2 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 4.3.2.2.3, “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 10-57.



**Figure 10-57. FCM Buffer RAM Memory Map During Boot Loading**

The process for booting is as follows:

1. Following negation of  $\overline{\text{PORESET}}$ , 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  $\text{FBAR}[\text{BLK}]$  points to the boot block.

4. 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  $\text{FMR}[\text{ECCM}] = 0$  for small-page devices, and  $\text{FMR}[\text{ECCM}] = 1$  for large-page devices.

If RCW initialization is required, the first 64 bytes of the boot block must be prepared in accordance with the layout described in [Section 4.3.3.1.1, “Local Bus Controller Setting.”](#)

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  $\overline{\text{hreset\_req}}$  signal.

- 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.

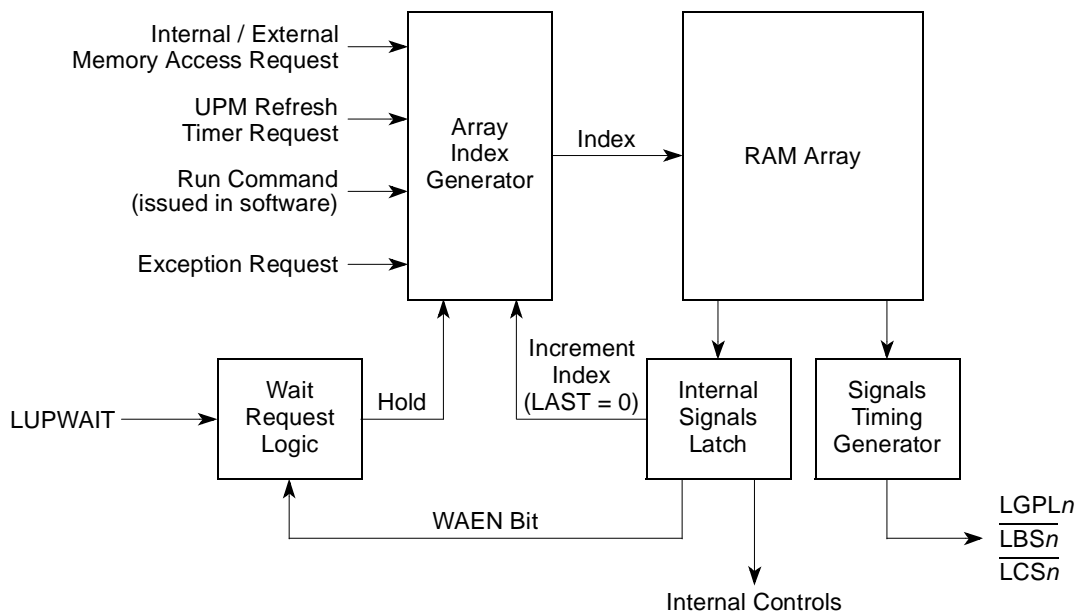
#### 10.4.4 User-Programmable Machines (UPMs)

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 ( $\overline{LCSn}$ ,  $\overline{LBS}[0:1]$  and  $\overline{LGPL}[0:5]$ ) 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  $\overline{LGPL4}/\overline{LGTA}/\overline{LFRB}/\overline{LUPWAIT}$  signal is used as both an input and an output, a weak pull-up is required. For details regarding termination options, see *PowerQUICC II Pro MPC8308 Hardware Specification*.

Figure 10-58 shows the basic operation of each UPM.



**Figure 10-58. 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.

#### 10.4.4.1 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 ( $MxMR[OP] = 00$ ):

- 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 10-59 and Table 10-39 show the start addresses of these patterns in the UPM RAM, according to cycle type. RUN commands ( $MxMR[OP] = 11$ ), however, can initiate patterns starting at any of the 64 UPM RAM words.

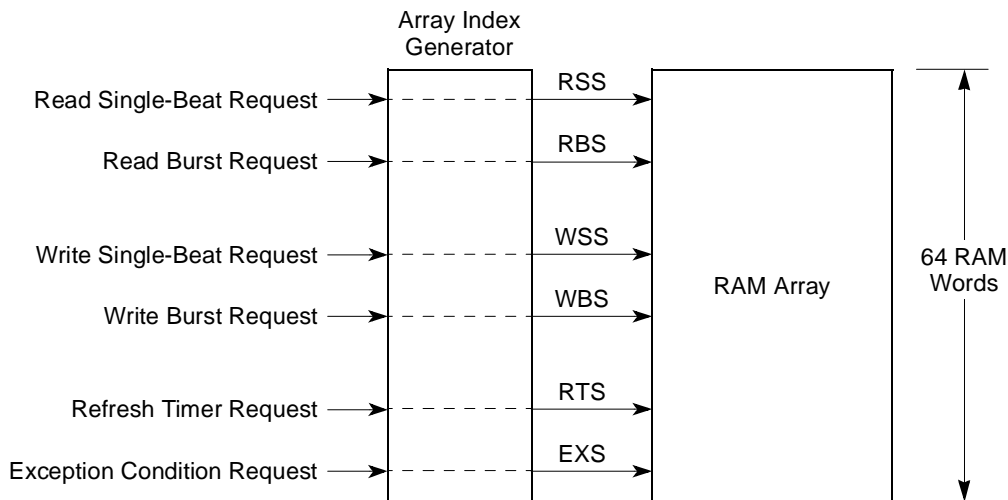


Figure 10-59. RAM Array Indexing

Table 10-39. 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

**Table 10-39. UPM Routines Start Addresses (continued)**

UPM Routine	Routine Start Address
Refresh timer (RTS)	0x30
Exception condition (EXS)	0x3C

#### 10.4.4.1.1 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:

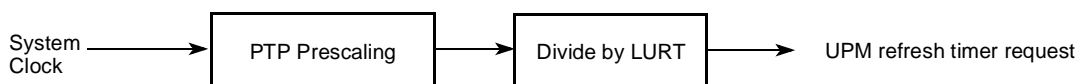
- A single-beat transfer transfers one operand consisting of up to a single word (dependent on port size). A single-beat cycle starts with one transfer start and ends with one transfer acknowledge.
- A burst transfer transfers exactly 4 double words regardless of port size. For 32-bit accesses, the burst 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  $ORn[BI]$ . 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.

#### 10.4.4.1.2 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 10-60](#) 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 10-60. 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.

### 10.4.4.1.3 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  $M_xMR[OP] = 11$  and accessing UPM $n$  memory region with any write transaction that hits the corresponding UPM machine.  $M_xMR[MAD]$  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 LD signals remain high-impedance unless a write occurs.

### 10.4.4.1.4 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.

### 10.4.4.2 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  $M_xMR$ .

Patterns are written to the RAM array by setting  $M_xMR[OP] = 01$  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  $M_xMR[OP] = 10$ ).

#### NOTE

$M_xMR$  / MDR registers should not be updated while dummy read/write access is still in progress. If the  $M_xMR[MAD]$  is incremented then the previous dummy transaction is already completed.

In order to enforce proper ordering between updates to the  $M_xMR$ /MDR register and the dummy accesses to the UPM memory region, two rules must be followed:

- Since the result of any update to the MxMR/MDR register must be in effect before the dummy read or write to the UPM region, a write to MxMR/MDR should be followed immediately by a read of MxMR/MDR.
- The UPM memory region should have the same MMU settings as the memory region containing the MxMR 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 MxMR. 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.

#### 10.4.4.2.1 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<sub>n</sub> and OR<sub>n</sub> registers have been previously set up:

1. Program MxMR 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 MxMR[MAD]. If incremented, the previous dummy write transaction is completed; proceed to step 6. Repeat step 5 until incremented.
6. Program MxMR for the second write with the desired RAM array address.
7. Write pattern/data to MDR to ensure that the MxMR 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 MxMR[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 MxMR to ensure that the MxMR has already been updated with the desired configuration.

#### 10.4.4.2.2 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 (MxMR[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  $MxMR$  for the first read with the desired RAM array address.
2. Read  $MxMR$  to ensure that the  $MxMR$  has already been updated with the desired configuration, such as RAM array address.
3. Perform a dummy read transaction.
4. Read/check  $MxMR[MAD]$ . If incremented, the previous dummy read transaction is completed; proceed to step 5. Repeat step 4 until incremented.
5. Read MDR.
6. Program  $MxMR$  for the second read with the desired RAM array address.
7. Read  $MxMR$  to ensure that the  $MxMR$  has already been updated with the desired configuration, such as RAM array address.
8. Perform a dummy read transaction.
9. Read/check  $MxMR[MAD]$ . If incremented, the previous dummy read transaction is completed; proceed to step 10. Repeat step 9 until incremented.
10. Read MDR.

### 10.4.4.3 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. For  $LCRR[CLKDIV] = 4$  or  $8$ , each bit in the RAM word relating to  $\overline{LCS}_n$  and  $\overline{LBS}_n$  timing specifies the value of the corresponding external signal at each quarter phase of the bus clock. If  $LCRR[CLKDIV] = 2$ , the external signal can change value only on each half phase of the bus clock. If the RAM word in this case ( $LCRR[CLKDIV] = 2$ ) specifies a quarter phase signal change, the signal timing generator interprets this as a half cycle change.

The division of UPM bus cycles into phases is shown in [Figure 10-61](#) and [Figure 10-62](#). If  $LCRR[CLKDIV] = 2$ , the bus cycle comprises only two active phases, T1 and T3, which correspond with the first and second halves of the bus clock cycle, respectively. However, if  $LCRR[CLKDIV] = 4$  or  $8$ , four phases, T1–T4, define four quarters of the bus clock cycle. Because T2 and T4 are inactive when  $LCRR[CLKDIV] = 2$ , UPM ignores signal timing programmed for assertion in either of these phases in the case  $LCRR[CLKDIV] = 2$ .

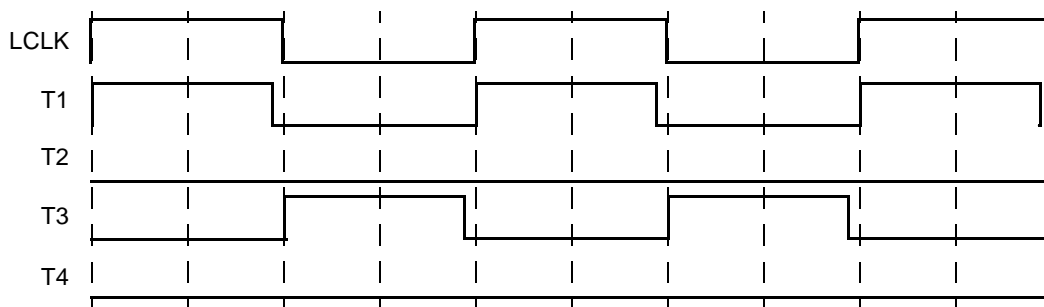
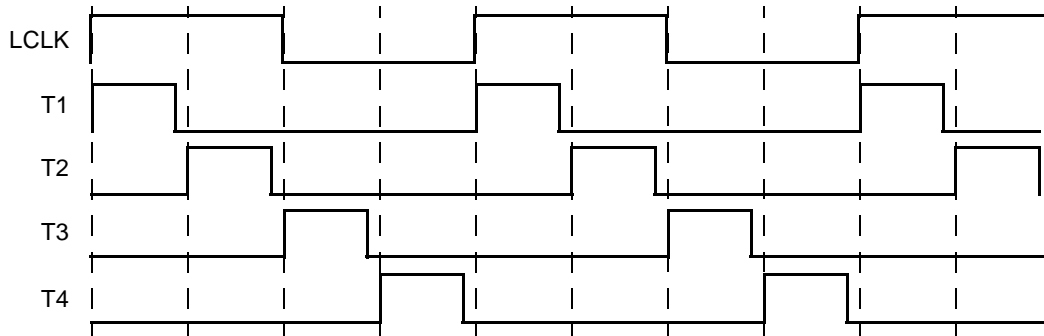


Figure 10-61. UPM Clock Scheme for  $LCRR[CLKDIV] = 2$

Figure 10-62. UPM Clock Scheme for  $LCRR[CLKDIV] = 4$  or  $8$ 

#### 10.4.4.4 RAM Array

The RAM array for each UPM is 64 locations deep and 32 bits wide, as shown in Figure 10-63. The signals at the bottom of the figure are UPM outputs. The selected  $\overline{LCS}_n$  is for the bank that matches the current address. The selected  $\overline{LBS}$  is for the byte lanes read or written by the access.

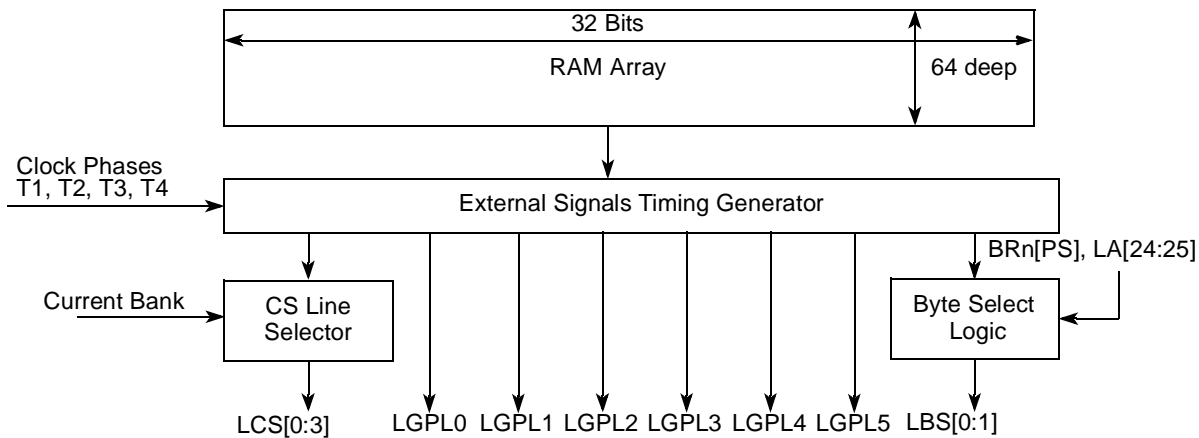


Figure 10-63. RAM Array and Signal Generation

##### 10.4.4.4.1 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 10-64 shows the RAM word fields. When  $LCRR[CLKDIV] = 4$  or  $8$ , the  $CST_n$  and  $BST_n$  bits determine the state of UPM signals  $\overline{LCS}_n$  and  $\overline{LBS}[0:1]$  at each quarter phase of the bus clock. When  $LCRR[CLKDIV] = 2$ ,  $CST_2$  and  $CST_4$  are ignored and the external has the values defined by  $CST_1$  and  $CST_3$  but extended to half the clock cycle in duration. The same interpretation occurs for the  $BST_n$  bits when  $LCRR[CLKDIV] = 2$ .

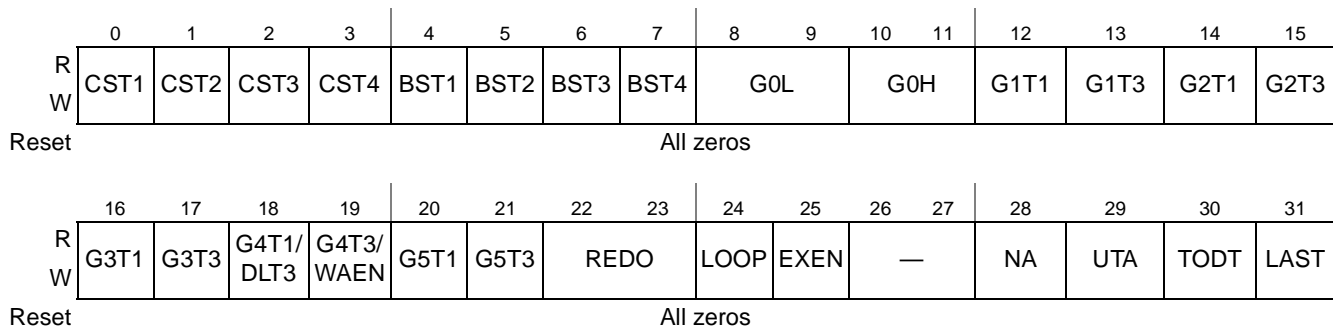


Figure 10-64. RAM Word Fields

Table 10-40 contains descriptions of the RAM word fields.

Table 10-40. RAM Word Field Descriptions

Bits	Name	Description
0	CST1	Chip select timing 1. Defines the state (0 or 1) of $\overline{LCSn}$ during bus clock quarter phase 1 if LCRR[CLKDIV] = 4 or 8. Defines the state (0 or 1) of $\overline{LCSn}$ during bus clock half phase 1 if LCRR[CLKDIV] = 2.
1	CST2	Chip select timing 2. Defines the state (0 or 1) of $\overline{LCSn}$ during bus clock quarter phase 2 if LCRR[CLKDIV] = 4 or 8. Ignored when LCRR[CLKDIV] = 2.
2	CST3	Chip select timing 3. Defines the state (0 or 1) of $\overline{LCSn}$ during bus clock quarter phase 3 if LCRR[CLKDIV] = 4 or 8. Defines the state (0 or 1) of $\overline{LCSn}$ during bus clock half phase 2 if LCRR[CLKDIV] = 2.
3	CST4	Chip select timing 4. Defines the state (0 or 1) of $\overline{LCSn}$ during bus clock quarter phase 4 if LCRR[CLKDIV] = 4 or 8. Ignored when LCRR[CLKDIV] = 2.
4	BST1	Byte select timing 1. Defines the state (0 or 1) of $\overline{LBS}$ during bus clock quarter phase 1 (LCRR[CLKDIV] = 4 or 8) or bus clock half phase 1 (LCRR[CLKDIV] = 2), in conjunction with BR $\eta$ [PS] and LA[24:25].
5	BST2	Byte select timing 2. Defines the state (0 or 1) of $\overline{LBS}$ during bus clock quarter phase 2 (LCRR[CLKDIV] = 4 or 8), in conjunction with BR $\eta$ [PS] and LA[24:25]. Ignored when LCRR[CLKDIV] = 2.
6	BST3	Byte select timing 3. Defines the state (0 or 1) of $\overline{LBS}$ during bus clock quarter phase 3 (LCRR[CLKDIV] = 4 or 8) or bus clock half phase 2 (LCRR[CLKDIV] = 2), in conjunction with BR $\eta$ [PS] and LA[24:25].
7	BST4	Byte select timing 4. Defines the state (0 or 1) of $\overline{LBS}$ during bus clock quarter phase 4 (LCRR[CLKDIV] = 4 or 8), in conjunction with BR $\eta$ [PS] and LA[24:25]. Ignored when LCRR[CLKDIV] = 2.
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). 00 Value defined by MxMR[G0CL] 01 Reserved 10 0 11 1

**Table 10-40. RAM Word Field Descriptions (continued)**

Bits	Name	Description
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). 00 Value defined by MxMR[G0CL] 01 Reserved 10 0 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).
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 MxMR[GPL4]. If MxMR[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). If MxMR[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: 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. 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 MxMR[GPL4]. If MxMR[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). If MxMR[GPL4] = 1 and LGPL4/LUPWAIT functions as an input (LUPWAIT), G4T3/WAEN is used to enable the wait mechanism: 0 LUPWAIT detection is disabled. 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. 00 Once (normal operation) 01 Twice 10 Three times 11 Four times



Table 10-40. RAM Word Field Descriptions (continued)

Bits	Name	Description
24	LOOP	<p>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 MxMR.</p> <p>0 The current RAM word is not the loop start word or loop end word. 1 The current RAM word is the start or end of a loop.</p>
25	EXEN	<p>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.</p> <p>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.</p> <p>0 The UPM continues executing the remaining RAM words, ignoring any internal bus monitor time-out. 1 The current RAM word allows a branch to the exception pattern after the current cycle if an exception condition is detected.</p>
26–27	—	Reserved
28	NA	<p>Next burst address. Determines when the address is incremented during a burst access.</p> <p>0 The address increment function is disabled. 1 The address is incremented in the next cycle. In conjunction with the BRn[PS], the increment value of LAN is 1 or 2 for port sizes of 8 and 16 bits, respectively.</p>
29	UTA	<p>UPM transfer acknowledge. Indicates assertion of transfer acknowledge in the current cycle.</p> <p>0 Transfer acknowledge is not asserted in the current cycle. 1 Transfer acknowledge is asserted in the current cycle.</p> <p>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.</p>
30	TODT	<p>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 MxMR[DSn]. The disable timer does not affect memory accesses to different banks. Note that TODT must be set together with LAST, otherwise it is ignored.</p> <p>0 The disable timer is turned off. 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.</p>
31	LAST	<p>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 MxMR[DSn].</p> <p>0 The UPM continues executing RAM words. 1 Indicates the last RAM word in the program. The service to the UPM request is done after this cycle concludes.</p> <p>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.</p>

#### 10.4.4.4.2 Chip-Select Signal Timing (CST<sub>n</sub>)

If  $BR_n[MSEL]$  of the accessed bank selects a UPM on the currently requested cycle, the UPM manipulates the  $\overline{LCS}_n$  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  $\overline{LCS}_n$  signal. The state of the selected  $\overline{LCS}_n$  signal of the corresponding bank depends on the value of each  $CST_n$  bit. Figure 10-65 shows how UPMs control  $\overline{LCS}_n$  signals.

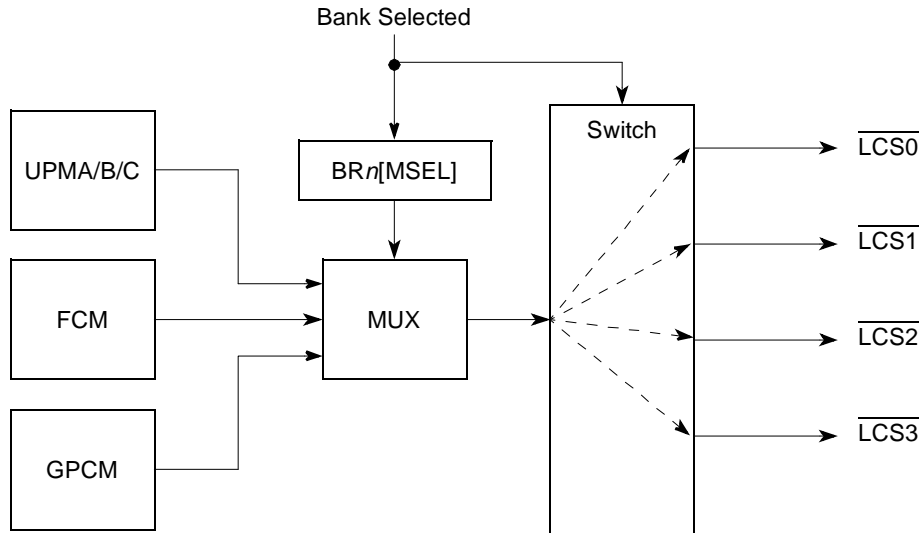


Figure 10-65.  $\overline{LCS}_n$  Signal Selection

#### 10.4.4.4.3 Byte Select Signal Timing (BST<sub>n</sub>)

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  $\overline{LBS}[0:1]$  signal. The timing of both byte-select signals is specified in the RAM word. However,  $\overline{LBS}[0:1]$  are also controlled by the port size of the accessed bank, the number of bytes to transfer, and the address accessed. Figure 10-66 shows how UPMs control  $\overline{LBS}[0:1]$ .

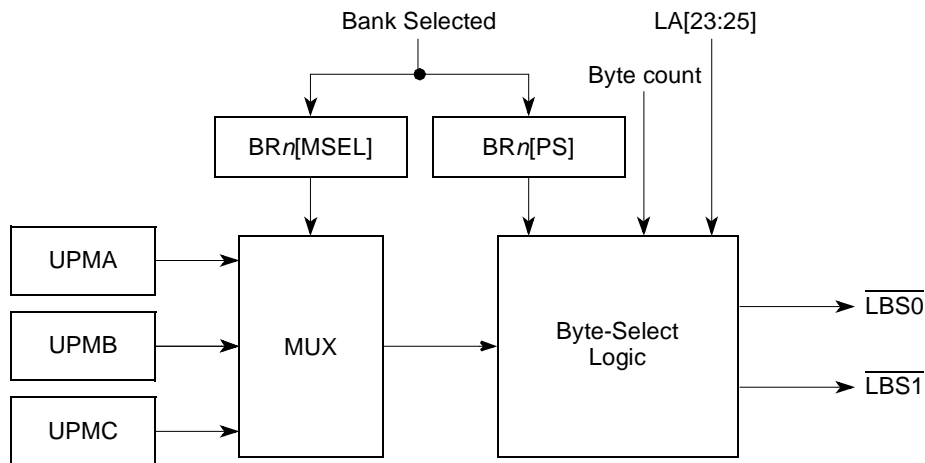


Figure 10-66.  $\overline{LBS}$  Signal Selection

The uppermost byte select ( $\overline{\text{LBS0}}$ ), when asserted, indicates that LD[0:7] contains valid data during a cycle. Likewise,  $\overline{\text{LBS1}}$  indicates that LD[8:15] contain valid data. For a UPM refresh timer request, all  $\overline{\text{LBS}}[0:1]$  signals are asserted/negated by the UPM according to the refresh pattern only. Following any internal bus monitor exception, the  $\overline{\text{LBS}}[0:1]$  signals are negated regardless of the exception handling provided by any UPM exception pattern to prevent spurious writes to external RAM.

#### 10.4.4.4.4 General-Purpose Signals ( $GnTn$ , $GOn$ )

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 MxMR[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.

#### 10.4.4.4.5 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 10-41. 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 that LAST and LOOP must not be set together.

**Table 10-41. MxMR 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

#### 10.4.4.4.6 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.

#### 10.4.4.4.7 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 MxMR[GPL4], determines when the data input is sampled by the eLBC as follows:

- If MxMR[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 MxMR[GPL4] = 0 (G4T4/DLT3 functions as G4T4), or if MxMR[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 10-67 shows how data sampling is controlled by the UPM.

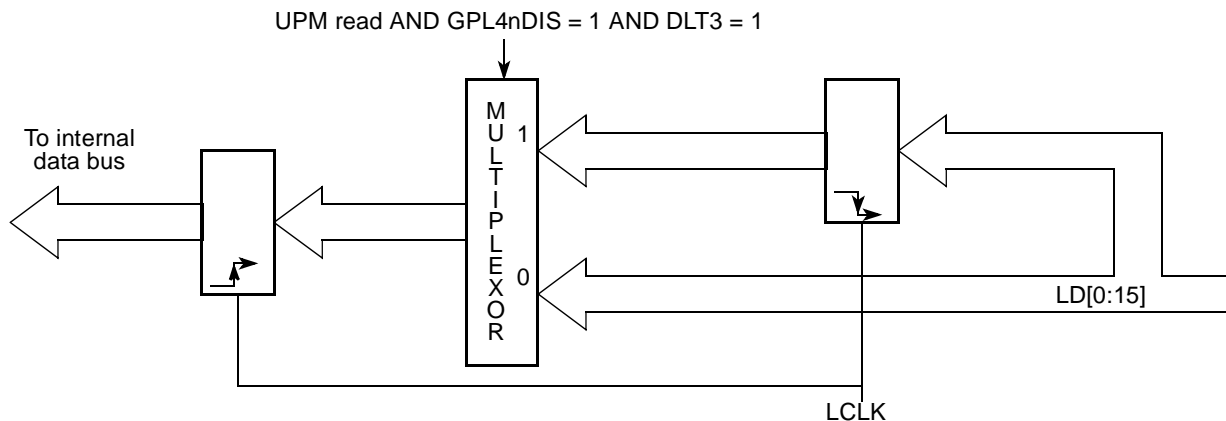


Figure 10-67. UPM Read Access Data Sampling

#### 10.4.4.4.8 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.

#### 10.4.4.4.9 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 10-68 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  $\overline{\text{LCSn}}$  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.

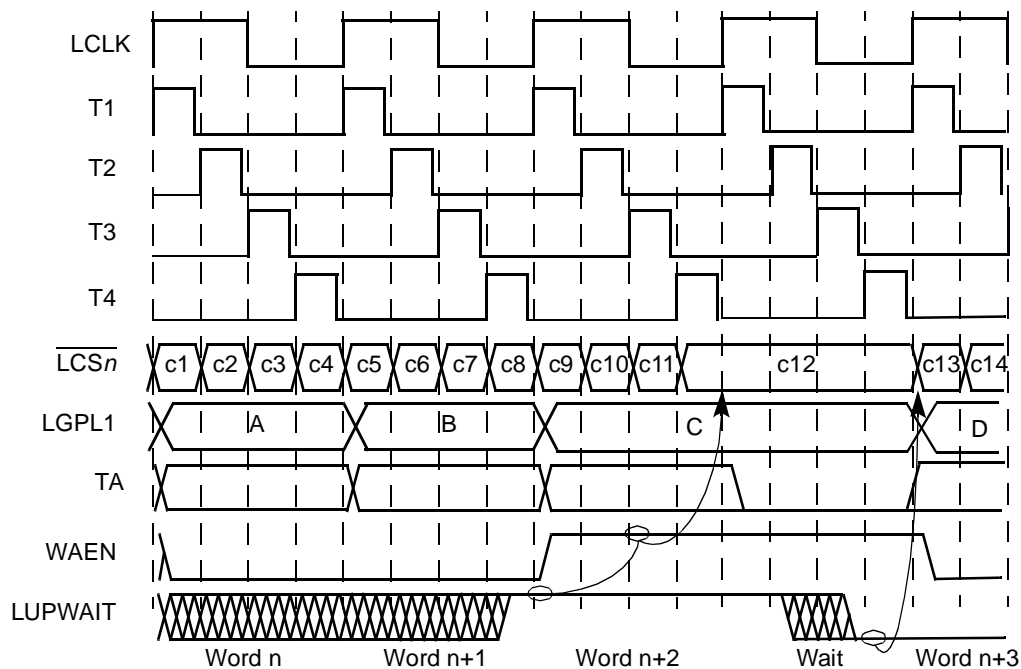


Figure 10-68. Effect of LUPWAIT Signal

#### 10.4.4.5 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 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. 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 in WAIT 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.

#### 10.4.4.6 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.

### 10.5 Initialization/Application Information

#### 10.5.1 Interfacing to Peripherals in Different Address Modes

This section provides guidelines for interfacing to peripherals.

##### 10.5.1.1 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.

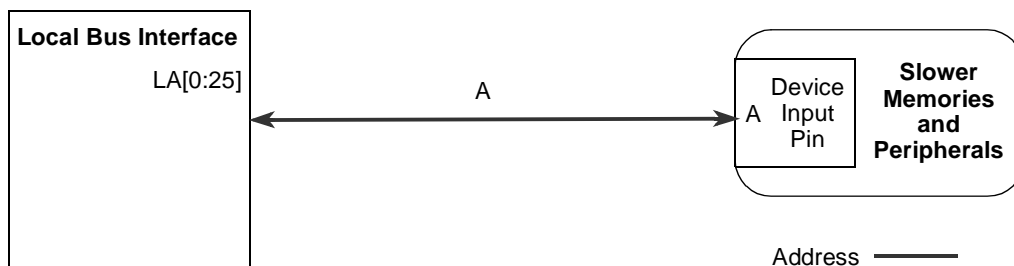


Figure 10-69. GPCM Address Timings

To calculate address setup timing for a slower peripheral/memory device, the address setup for the actual peripheral needs to be added.

For data timings, only the propagation delay of one buffer plus the actual data setup time has to be considered.

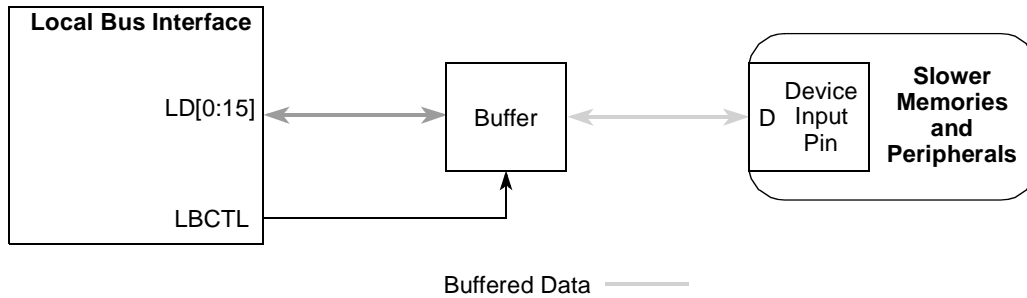


Figure 10-70. GPCM Data Timings

## 10.5.2 Interface to Different Port-Size Devices

The eLBC supports 8- and 16-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 16-bit port must reside on LD[0:15], and an 8-bit port must reside on LD[0:7]. The local bus always tries to transfer the maximum amount of data on all bus cycles. Figure 10-71 shows the device connections on the data bus.

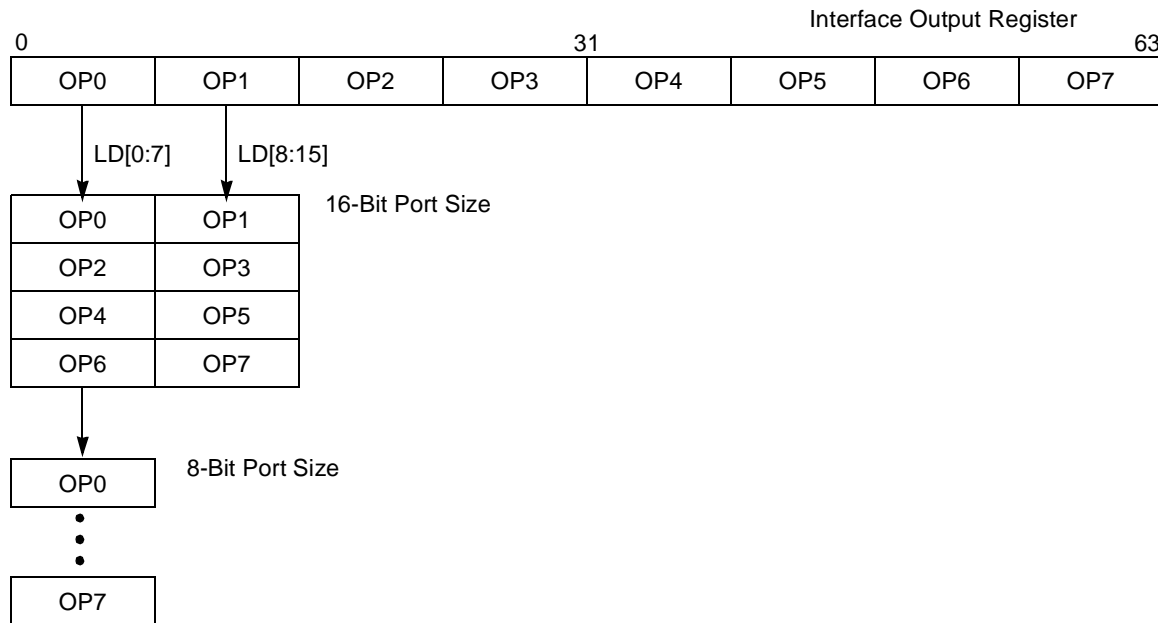


Figure 10-71. Interface to Different Port-Size Devices

Table 10-42 lists the bytes required on the data bus for read cycles.

**Table 10-42. Data Bus Drive Requirements For Read Cycles**

Transfer Size	Address State <sup>1</sup> 3 lsbs	Port Size/LD Data Bus Assignments							
		16-Bit				8-Bit			
		0-7	8-15	16-23	24-31	0-7	8-15	16-23	24-31
Byte	000	OP0	—			OP0			
	001	—	OP1			OP1			
	010	OP2	—			OP2			
	011	—	OP3			OP3			
	100	OP4	—			OP4			
	101	—	OP5			OP5			
	110	OP6	—			OP6			
	111	—	OP7			OP7			
Half Word	000	OP0	OP1			OP0			
	001	—	OP1			OP1			
	010	OP2	OP3			OP2			
	100	OP4	OP5			OP4			
	101	—	OP5			OP5			
	110	OP6	OP7			OP6			
Word	000	OP0	OP1			OP0			
	100	OP4	OP5			OP4			

<sup>1</sup> Address state is the calculated address for port size.

### 10.5.3 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 ( $BRn[MSEL] = 001$ ), with base address, port size, ECC mode, and timing parameters configured in accordance with the device's hardware specifications.



### 10.5.3.1 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 10-43](#). 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 (LTERSR[CC]) if interrupts are enabled.

**Table 10-43. FCM Register Settings for Soft Reset (OR<sub>n</sub>[PGS] = 1)**

Register	Initial Contents	Description
FCR	0xFF000000	CMD0 = 0xFF = reset command; other commands unused
FBAR	—	unused
FPAR	—	unused
FBCR	—	unused
MDR	—	unused
FIR	0x40000000	OP0 = CM0 = command 0; OP1–OP7 = NOP

### 10.5.3.2 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 10-44](#). 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 (LTERSR[CC]) if interrupts are enabled.

**Table 10-44. FCM Register Settings for Status Read (OR<sub>n</sub>[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; OP1 = RS = read status to MDR; OP2–OP7 = NOP

### 10.5.3.3 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 10-45](#). 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 10-45. FCM Register Settings for ID Read (ORn[PGS] = 1)**

Register	Initial Contents	Description
FCR	0x90000000	CMD0 = 0x90 = read ID command; other commands unused
FBAR	—	unused
FPAR	—	unused
FBCR	—	unused
MDR	0x00000000	AS0 = 0x00 = dummy address for read ID command; AS0–AS3 return with first 4 bytes of ID code
FIR	0x43BBBBB0	OP0 = CM0 = command 0; OP1 = UA = user address from MDR; OP2–OP6 = RS = read 4 bytes ID into MDR[AS3–AS0]; OP7 = NOP

#### 10.5.3.4 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 10-46](#). 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 10-46. FCM Register Settings for Page Read (ORn[PGS] = 1)**

Register	Initial Contents	Description
FCR	0x00300000	CMD0 = 0x00 = random read address entry; CMD1 = 0x30 = read page
FBAR	block index (for example, block 0x00010ab4)	BLK locates index of 128-Kbyte block
FPAR	page offset (for example, 0x00005000 locates page 5, buffer 1)	PI locates page index in BLK; PI mod 2 indexes FCM buffer RAM; MS = 0 and CI = 0
FBCR	0x00000000	BC = 0 to read entire 2112-byte page with ECC check

**Table 10-46. FCM Register Settings for Page Read (OR<sub>n</sub>[PGS] = 1) (continued)**

Register	Initial Contents	Description
MDR	—	unused
FIR	0x4125E000	OP0 = CM0 = command 0; OP1 = CA = column address; OP2 = PA = page address; OP3 = CM1 = command 1; OP4 = RBW = wait on Flash ready and read data into FCM buffer; OP5–OP7 = NOP

### 10.5.3.5 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 10-47](#). 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 10-47. FCM Register Settings for Block Erase (OR<sub>n</sub>[PGS] = 1)**

Register	Initial Contents	Description
FCR	0x6070D000	CMD0 = 0x60 = block address entry; CMD1 = 0x70 = read status CMD2 = 0xD0 = erase block;
FBAR	block index (for example, 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; OP1 = PA = page address; OP2 = CM2 = command 2; OP3 = CW1 = wait on Flash ready and issue command 1; OP4 = RS = read erase status into MDR[AS0]; OP5–OP7 = NOP

### 10.5.3.6 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 10-48](#). 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 10-48. FCM Register Settings for Page Program (ORn[PGS] = 1)**

Register	Initial Contents	Description
FCR	0x80701000	CMD0 = 0x80 = page address and data entry; CMD1 = 0x70 = read status CMD2 = 0x10 = program page;
FBAR	block index (for example, block 0x00010AB4)	BLK locates index of 128-Kbyte block
FPAR	page offset (for example, 0x00005000 locates page 5, buffer 1)	PI locates page index in BLK; PI mod 2 indexes FCM buffer RAM; 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; OP1 = CA = column address; OP2 = PA = page address; OP3 = WB = write data from buffer; OP4 = CM2 = command 2; OP5 = CW1 = wait on Flash ready and issue command 1; OP6 = RS = read erase status into MDR[AS0]; OP7 = NOP

### 10.5.4 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 (or 8). These illustrative examples may not represent the timing necessary for any

specific device used with the eLBC. Here, LGPL1 is programmed to drive  $\overline{R/\overline{W}}$  of the DRAM, although any  $\text{LGPL}_n$  signal may be used for this purpose.

Figure 10-72 shows single-beat read access to FPM DRAM.

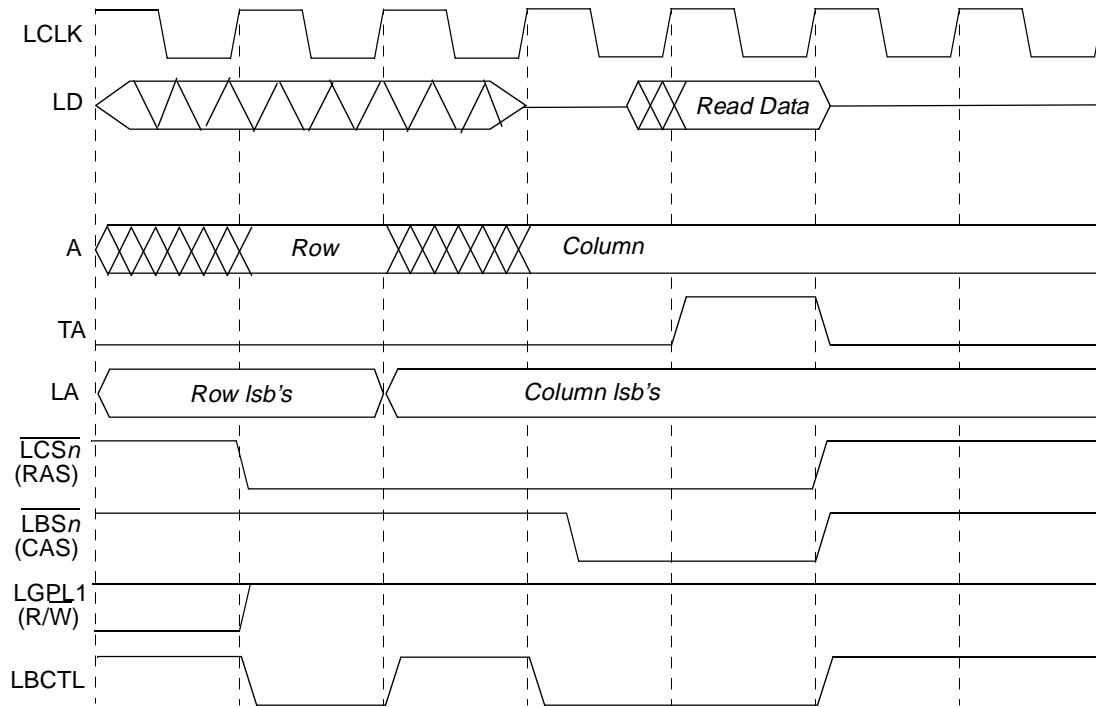


Figure 10-72. Single-Beat Read Access to FPM DRAM

Table 10-49 lists UPM code for single-beat read access.

Table 10-49. UPM Code for Single-Beat Read Access

cst1	0	LA 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

Table 10-49. UPM Code for Single-Beat Read Access (continued)

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 10-73 shows single-beat write access to FPM DRAM.

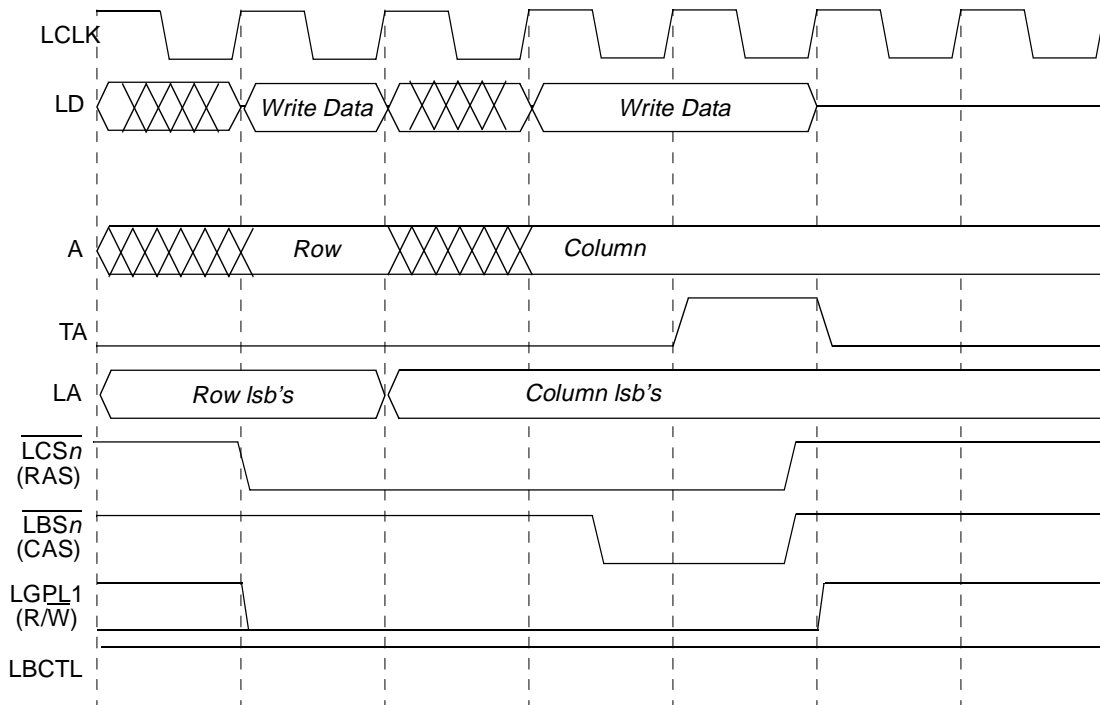


Figure 10-73. Single-Beat Write Access to FPM DRAM

Table 10-50 lists UPM code for single-beat write access.

Table 10-50. UPM Code for Single-Beat Write Access

cst1	0	LA 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

Table 10-50. UPM Code for Single-Beat Write Access (continued)

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 10-74 shows burst read access write access to FPM DRAM using LOOP.

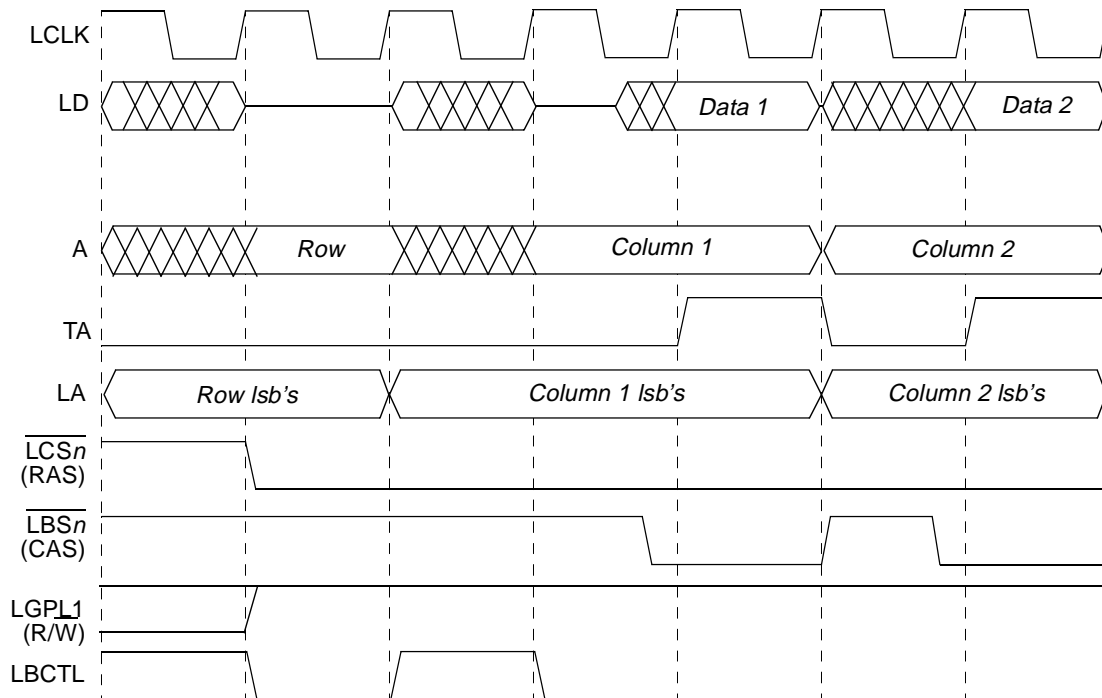


Figure 10-74. Burst Read Access to FPM DRAM Using LOOP (Two Beats)

Table 10-51 lists UPM code for burst read access.

Table 10-51. UPM Code for Burst Read Access

cst1	0	LA 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

Table 10-51. UPM Code for Burst Read Access (continued)

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 10-75 shows refresh cycle (CBR) to FPM DRAM.

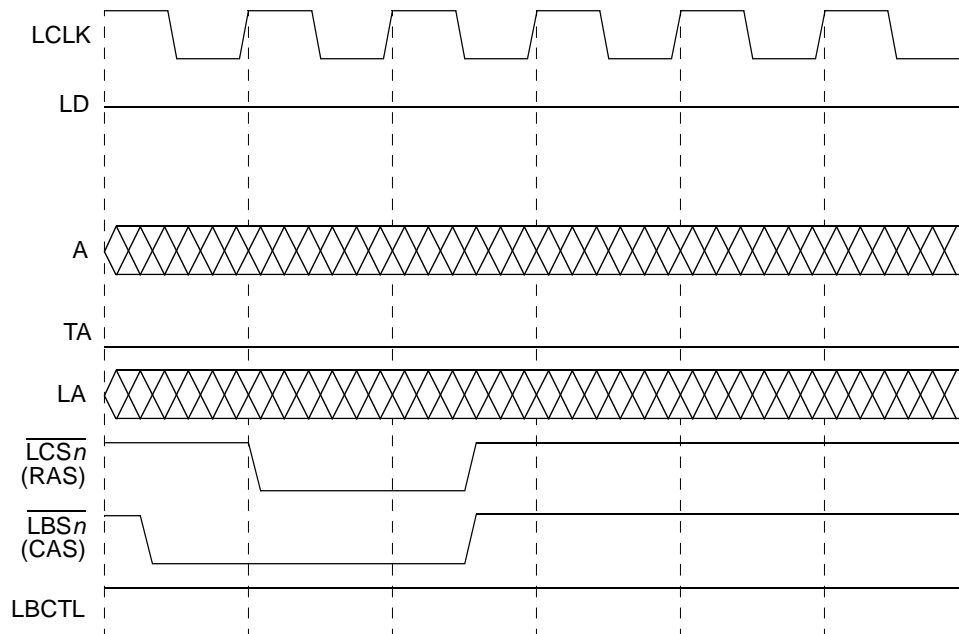


Figure 10-75. Refresh Cycle (CBR) to FPM DRAM

Table 10-52 lists UPM code for refresh cycle.

Table 10-52. UPM Code for Refresh Cycle

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

Table 10-52. UPM Code for Refresh Cycle (continued)

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
	PTS	PTS + 1	PTS + 2	

Figure 10-76 shows exception cycle.

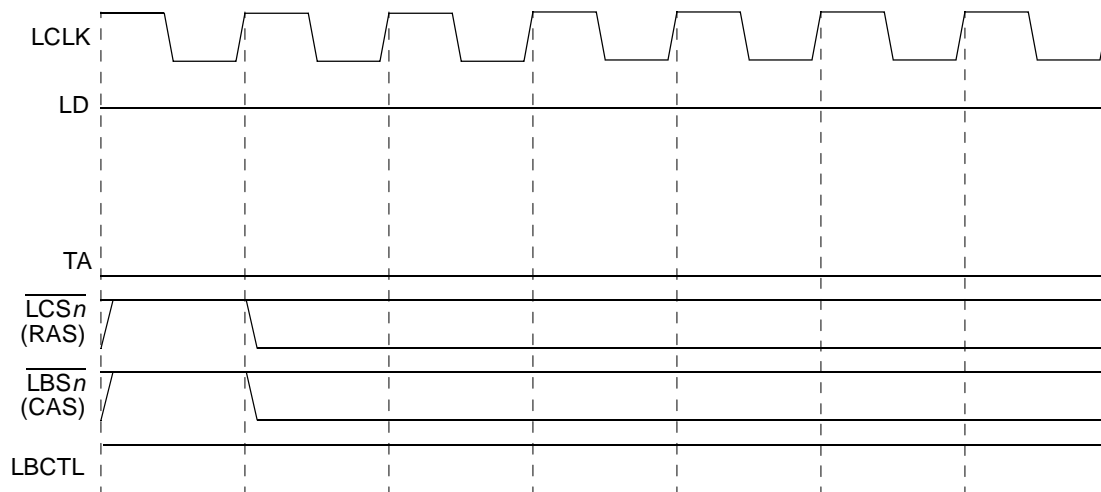


Figure 10-76. Exception Cycle

Table 10-53 lists UPM code for exception cycle.

**Table 10-53. UPM Code for Exception Cycle**

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	

## 10.5.5 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 10-77 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. ZBT SRAMs are mostly used by performance-critical applications, therefore, it is assumed that, typically, the maximum width of the local bus of 16 bits will be used.

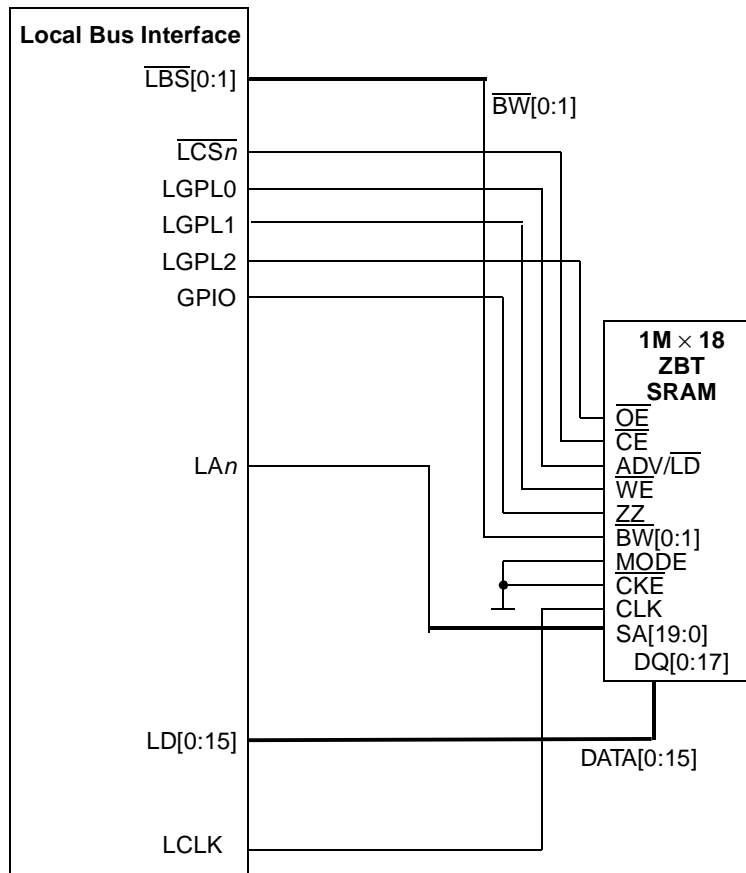


Figure 10-77. Interface to ZBT SRAM

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.  $\overline{\text{CKE}}$  should also be tied to ground.

ZBT SRAMs perform four-beat bursts. Because the eLBC generates sixteen-beat transactions (for 16-bit ports) the UPM breaks down each burst into four consecutive four-beat bursts. The internal address generator of the eLBC generates the new  $\{A21, A22\}$  for the second, third, and fourth 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  $[0:1:2:3:4:5:6:7:\dots:15]$ . Therefore, four consecutive linear bursts of the ZBT SRAM with  $\{A21, A22\} = \{0,0\}$  for the first burst,  $\{A21, A22\} = \{0,1\}$  for the second burst,  $\{A21, A22\} = \{1,0\}$  for the third burst, and  $\{A21, A22\} = \{1,1\}$  for the fourth 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  $\overline{WE}$ ). The UPM controller basically has to wait for the end of the SRAM burst to avoid bus contention with further bus activities.

If a UPM device has  $\overline{OE}$ , it should not be asserted in the same RAM word as the TA signal. If  $\overline{OE}$  and TA are both asserted in the same RAM word, then the eLBC may not be able to sample the correct data during reads. Therefore  $\overline{OE}$  must be asserted earlier than TA.





# Chapter 11

## Enhanced Secure Digital Host Controller

### 11.1 Overview

The enhanced secure digital host controller (eSDHC) provides an interface between the host system and these types of memory cards:

- Multi Media card (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 MMCs or downloadable from cellular phones, WLAN, or other wireless networks.

- 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.

- SDIO

Under the SD protocol, the SD cards 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. The I/O card provides high-speed data I/O with low power consumption for mobile electronic devices. The combo card has both memory and I/O functions. To keep

[Figure 11-1](#) simple, the diagram does not show cards with reduced sizes.

The eSDHC acts as a bridge, passing host bus transactions to SD card/SDIO/MMCs by sending commands and performing data accesses to or from the cards. It handles the SD/SDIO/MMC protocol at the transmission level. [Figure 11-1](#) shows connection of the eSDHC.

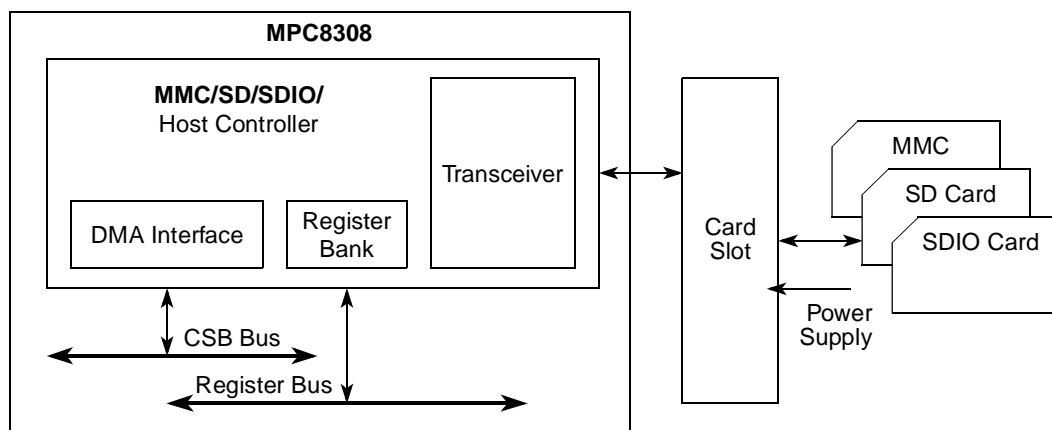


Figure 11-1. System Connection of the eSDHC

Figure 11-2 is a block diagram of the eSDHC.

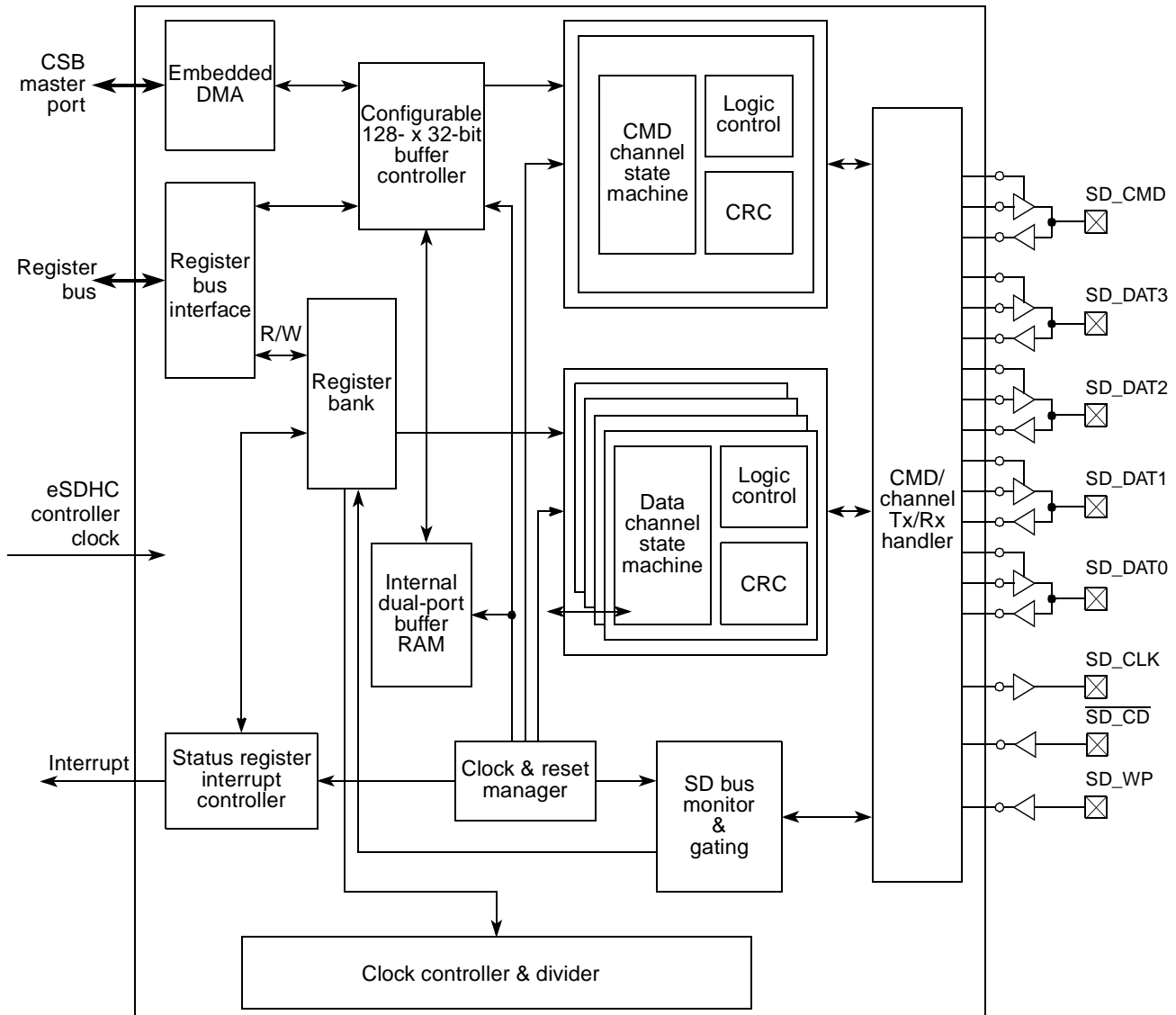


Figure 11-2. eSDHC Block Diagram

## 11.2 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
  - *MultiMedia Card System Specification, Version 4.0* (<http://www.mmca.org>)
  - *SD Memory Card Specification, Version 2.0* (<http://www.sdcard.org>)
  - *SD Specifications, Part 1, Physical Layer Specification, Version 2.0*

- *SD Card Specification, Part E1, SD Input/Output (SDIO) Card Specification, Version 2.0*
- Designed to work with SD Memory, miniSD Memory, SDIO, miniSDIO, SD Combo, SDIO, MMC, MMC*plus*, and RS-MMCs
- Card bus clock frequency up to 50 MHz
- Supports 1-/4-bit SD and SDIO modes, 1-/4-bit MMC modes
  - Up to 200 Mbps data transfer for SD cards/SDIO/MMCs using four parallel data lines
- Single- and multi-block read and write
- Supports block sizes of 1 ~ 4096 bytes
- Write-protection switch for write operations
- Synchronous abort
- Pause during the data transfer at a block gap
- SDIO read wait and suspend/resume operations
- Auto CMD12 for multi-block transfer
- Host can initiate non-data transfer commands while the data transfer is in progress
- Allows cards to interrupt the host in 1- and 4-bit SDIO modes
- Supports interrupt period, defined in the SDIO standard
- Fully configurable 128 × 32-bit FIFO for read/write data
- DMA capabilities

### 11.2.1 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
- Identification mode (upto 400 KHz)
- Full-speed mode (up to 25 MHz) or high-speed mode (up to 50 MHz)

### 11.3 External Signal Description

The eSDHC has eight chip I/O signals.

- SD\_CLK is the internally generated clock signal that drives the MMC, SDIO, or SD card.
- SD\_CMD I/O sends commands and receives responses from the card.
- SD\_DAT3–SD\_DAT0 performs data transfers between the eSDHC and the card.
- $\overline{\text{SD\_CD}}$  and  $\overline{\text{SD\_WP}}$  are card detection and write protection signals from the socket.
  - Signals  $\overline{\text{SD\_CD}}$  and  $\overline{\text{SD\_WP}}$  are optional for system implementation.

Table 11-1 shows the properties of the eSDHC I/O signals.

**Table 11-1. Signal Properties**

Name	Port	Function	Reset State	Pull up/Pull down Required
SD_CLK	O	Clock for MMC/SD/SDIO card	0	N/A
SD_CMD	I/O	Command line to card	High impedance	Pull up
SD_DAT3	I/O	<b>4-bit mode:</b> DAT3 line or configured as card detection pin <b>1-bit mode:</b> May be configured as card detection pin	High impedance	Board should have 100-K pull down. The card drives 50-K pull up as required by the <i>SD Card Specification</i> .
SD_DAT2	I/O	<b>4-bit mode:</b> DAT2 line or read wait <b>1-bit mode:</b> Read wait	High impedance	Pull up
SD_DAT1	I/O	<b>4-bit mode:</b> DAT1 line or interrupt detect <b>1-bit mode:</b> Interrupt detect	High impedance	Pull up
SD_DAT0	I/O	DAT0 line or busy-state detect	High impedance	Pull up
$\overline{\text{SD\_CD}}$	I	Card detection pin <ul style="list-style-type: none"> <li><math>\overline{\text{SD\_CD}}</math> = HIGH implies card is not present</li> <li><math>\overline{\text{SD\_CD}}</math> = LOW implies card is present.</li> </ul>	N/A	Board should have 100-K pull up on the $\overline{\text{SD\_CD}}$ signal and a 330 $\Omega$ pull down on the common pin of the Write Protect/Card Detect switch on the SD card connector.
SD_WP	I	Card write protect detect; if not used, tied low <ul style="list-style-type: none"> <li>SD_WP = HIGH implies the write protect switch is enabled on the card (Writes disabled)</li> <li>SD_WP = LOW implies write protect switch is disabled on the card (Writes enabled)</li> </ul>	N/A	Board should have 100-K pull up on the SD_WP signal and a 330 $\Omega$ pull down on the common pin of the Write Protect/Card Detect switch on the SD card connector.

## 11.4 Memory Map/Register Definition

Table 11-2 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 IMMRBAR together with the eSDHC block base address and offset listed in Table 11-2. 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 11-2. eSDHC Memory Map

eSDHC Registers—Block Base Address 0x2_E000				
Offset	Register	Access	Reset	Section/Page
0x000	DMA system address (DSADDR)	R/W	0x0000_0000	<a href="#">11.4.1/11-6</a>
0x004	Block attributes (BLKATTR)	R/W	0x0001_0000	<a href="#">11.4.2/11-6</a>
0x008	Command argument (CMDARG)	R/W	0x0000_0000	<a href="#">11.4.3/11-7</a>
0x00C	Command transfer type (XFERTYP)	R/W	0x0000_0000	<a href="#">11.4.4/11-8</a>
0x010	Command response0 (CMDRSP0)	R	0x0000_0000	<a href="#">11.4.5/11-11</a>
0x014	Command response1 (CMDRSP1)	R	0x0000_0000	<a href="#">11.4.5/11-11</a>
0x018	Command response2 (CMDRSP2)	R	0x0000_0000	<a href="#">11.4.5/11-11</a>
0x01C	Command response3 (CMDRSP3)	R	0x0000_0000	<a href="#">11.4.5/11-11</a>
0x020	Data buffer access port (DATPORT)	R/W	0x0000_0000	<a href="#">11.4.6/11-13</a>
0x024	Present state (PRSTAT)	R	0x0F80_00F8	<a href="#">11.4.7/11-13</a>
0x028	Protocol control (PROCTL)	R/W	0x0000_0020	<a href="#">11.4.8/11-17</a>
0x02C	System control (SYSCTL)	Mixed	0x0000_8008	<a href="#">11.4.9/11-20</a>
0x030	Interrupt status (IRQSTAT)	w1c	0x0000_0000	<a href="#">11.4.10/11-23</a>
0x034	Interrupt status enable (IRQSTATEN)	R/W	0x117F_013F	<a href="#">11.4.11/11-27</a>
0x038	Interrupt signal enable (IRQSIGEN)	R/W	0x0000_0000	<a href="#">11.4.12/11-30</a>
0x03C	Auto CMD12 status (AUTOC12ERR)	R	0x0000_0000	<a href="#">11.4.13/11-32</a>
0x040	Host controller capabilities (HOSTCAPBLT)	R	0x01F3_0000	<a href="#">11.4.14/11-34</a>
0x044 <sup>1</sup>	Watermark level (WML)	R/W	0x1010_1010	<a href="#">11.4.15/11-35</a>
0x050	Force event (FEVT)	W	0x0000_0000	<a href="#">11.4.16/11-35</a>
0x0FC	Host controller version (HOSTVER)	R	0x0000_1201	<a href="#">11.4.17/11-37</a>
0x40C	DMA control register (DCR)	R/W	0x0000_0000	<a href="#">11.4.18/11-37</a>

<sup>1</sup> The addresses following 0x044, except 0x050, 0x0FC and 0x40C, are reserved and read as all 0s. Writes to these registers are ignored.

### NOTE

For details on programming the CSB/eSDHC interface, see [Section 5.2.2.12, “eSDHC Control Registers \(SDHCCR\).”](#)

### 11.4.1 DMA System Address Register (DSADDR)

The DMA system address register contains 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.

Figure 11-3 shows the DMA system address register.

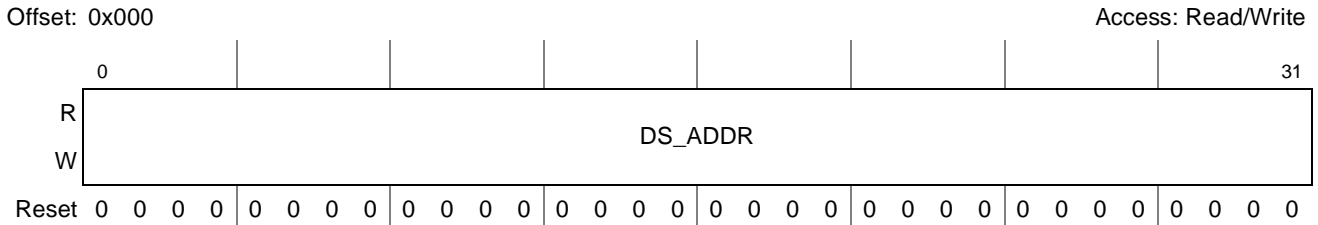


Figure 11-3. DMA System Address Register (DSADDR)

Table 11-3 describes the DSADDR fields.

Table 11-3. DSADDR Field Descriptions

Bit	Name	Description
0–31	DS_ADDR	DMA system address. When the eSDHC stops a DMA transfer, this register points to the system address of the next contiguous data position. <b>Note:</b> The DS_ADDR must be aligned to a four-byte boundary; the two least-significant bits must be cleared.

### 11.4.2 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, the following may occur:

- Reading this register may return an invalid value.
- Writing this register is ignored.

Figure 11-4 shows the DMA system address register.

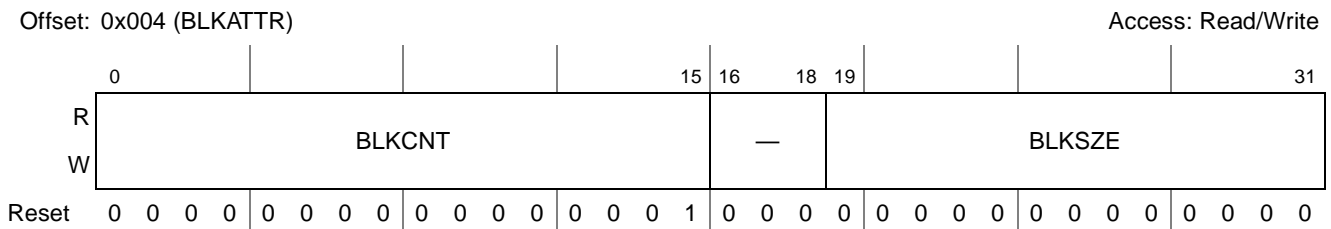


Figure 11-4. Block Attributes Register (BLKATTR)

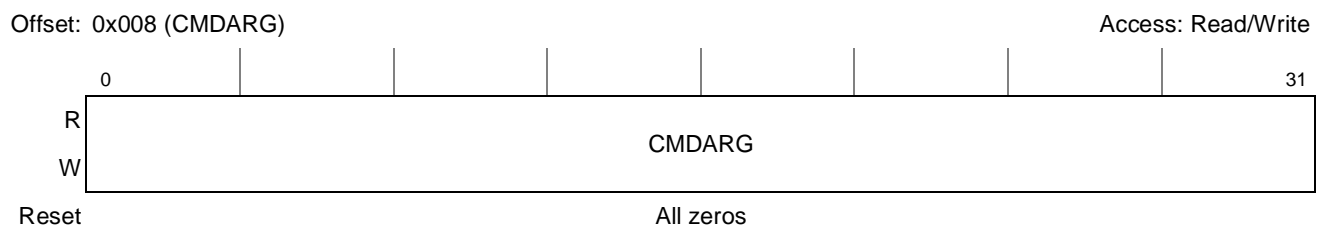
Table 11-4 describes the BLKATTR fields.

**Table 11-4. BLKATTR Field Descriptions**

Bit	Name	Description
0–15	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. 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. 0000 Stop count 0001 1 block 0002 2 blocks ... FFFF 65,535 blocks
16–18	—	Reserved
19–31	BLKSIZE	Transfer block size. Specifies the block size for block data transfers. Values can range from one byte up to the maximum buffer size. The DMA always writes at least four bytes to memory. Thus, software should allocate a buffer space rounded up to a 4-byte aligned size in order to avoid data corruption. 0000 No data transfer 0001 1 byte 0002 2 bytes 0003 3 bytes 0004 4 bytes ... 01FF 511 bytes 0200 512 bytes ... 0800 2048 bytes 1000 4096 bytes

### 11.4.3 Command Argument Register (CMDARG)

The command argument register, shown in Figure 11-5, contains the SD/MMC command argument.



**Figure 11-5. Command Argument Register (CMDARG)**

Table 11-5 describes the CMDARG fields.

**Table 11-5. CMDARG Field Descriptions**

Bit	Name	Description
0–31	CMDARG	Command argument. The SD/MMC command argument is specified as bits 39–8 of the command format in the <i>SD or MMC Specification</i> . If PRSSTAT[CMD] is set, this register is write-protected.

#### 11.4.4 Transfer Type Register (XFERTYP)

The transfer type register, shown in Figure 11-6, 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	—			CMDINX	CMD TYP	DP SEL	CICEN	CCCEN	—	RSP TYP			—		MSB SEL	DTD SEL		—	AC12 EN	BCEN	DMA EN
W																					

Reset

All zeros

**Figure 11-6. Transfer Type Register (XFERTYP)**

Table 11-6 describes the XFERTYP fields.

**Table 11-6. XFERTYP Field Descriptions**

Bit	Name	Description
0–1	—	Reserved
2–7	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 <i>SD Memory Card Physical Layer Specification</i> and <i>SDIO Card Specification</i> .



Table 11-6. XFERTYP Field Descriptions (continued)

Bit	Name	Description
8–9	CMDTYP	<p>Command type. There are three types of special commands: suspend, resume, and abort. Clear this bit field for all other commands.</p> <ul style="list-style-type: none"> <li>• Suspend command. 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 SD_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. 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.</li> <li>• 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.</li> <li>• Abort command. If this command is set when executing a read transfer, the eSDHC stops reads to the buffer. If this command is set when executing a write transfer, the eSDHC stops driving the SD_DAT line. After issuing the abort command, the host driver should issue a software reset. (Abort transaction)</li> </ul> <p>00 Normal—other commands 01 Suspend—CMD52 for writing bus suspend in the common card control register (CCCR), defined in the <i>SDIO Specification</i> 10 Resume—CMD52 for writing function select in CCCR 11 Abort—CMD12, CMD52 for writing I/O abort in CCCR</p>
10	DPSEL	<p>Data present select. Set to indicate that data is present and should be transferred using the SD_DAT line. It is cleared for the following:</p> <ul style="list-style-type: none"> <li>• Commands using only the SD_CMD line (e.g. CMD52)</li> <li>• Commands with no data transfer but using busy signal on the SD_DAT[0] line (R1b or R5b, e.g. CMD38)</li> </ul> <p><b>Note:</b> 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.</p> <p>0 No data present 1 Data present</p>
11	CICEN	<p>Command index check enable.</p> <p>0 Disable. The index field is not checked. 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.</p>
12	CCCEN	<p>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 <a href="#">Table 11-8</a>.)</p> <p>0 Disable. The CRC field is not checked. 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.</p>
13	—	Reserved

**Table 11-6. XFERTYP Field Descriptions (continued)**

Bit	Name	Description
14–15	RSPTYP	Response type select. 00 No response 01 Response length 136 10 Response length 48 11 Response length 48 check busy after response
16–25	—	Reserved
26	MSBSEL	Multi/single block select. Enables multiple block SD_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 <a href="#">Table 11-7</a> .) 0 Single block 1 Multiple blocks
27	DTDSEL	Data transfer direction select. Defines the direction of SD_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. 0 Write (host to card) 1 Read (card to host)
28	—	Reserved
29	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. 0 Disable 1 Enable
30	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. 0 Disable 1 Enable
31	DMAEN	DMA enable. Enables DMA functionality as described in <a href="#">Section 11.5.2, “DMA CSB Interface.”</a> If this bit is set, a DMA operation should begin when the host driver writes to the CMDINX field of the transfer type register. 0 Disable 1 Enable

[Table 11-7](#) shows how register settings determine types of data transfers.

**Table 11-7. 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 11-8 shows how the response type can be determined by the command index check enable, command CRC check enable, and response type bits.

**Table 11-8. Relation Between Parameters and Name of Response Type**

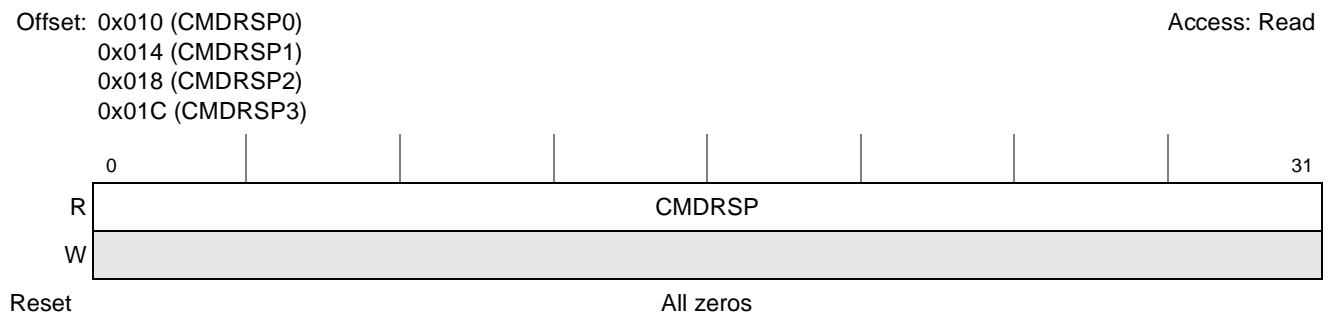
Response Type XFERTYP[RSPTYP]	Index Check Enable XFERTYP[CICEN]	CRC Check Enable XFERTYP[CCEN]	Response Type
00	0	0	No Response
01	0	1	R2
10	0	0	R3, R4
10	1	1	R1, R5, R6
11	1	1	R1b, R5b

#### NOTE

- In the *SDIO Specification*, response type notation of R5b is not defined. R5 includes R5b in the *SDIO Specification*. But R5b is defined to specify the eSDHC checks busy status after receiving a response. For example, usually CMD52 is used as R5 but the I/O abort command should be used as R5b.
- The CRC field for R3 and R4 is expected to be all 1s. The CRC check should be disabled for these response types.

### 11.4.5 Command Response 0–3 (CMDRSP0–3)

The command response registers, shown in Figure 11-7, store the four parts of the response bits from the card.



**Figure 11-7. Command Response 0–3 Register (CMDRSPn)**

Table 11-9 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 11-9. 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	R[39–8]	CMDRSP0
R5, R5b	SDIO response	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.

## 11.4.6 Buffer Data Port Register (DATPORT)

The buffer data port register, shown in Figure 11-8, 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 if the data will be written (or read) by the eSDHC internal DMA.

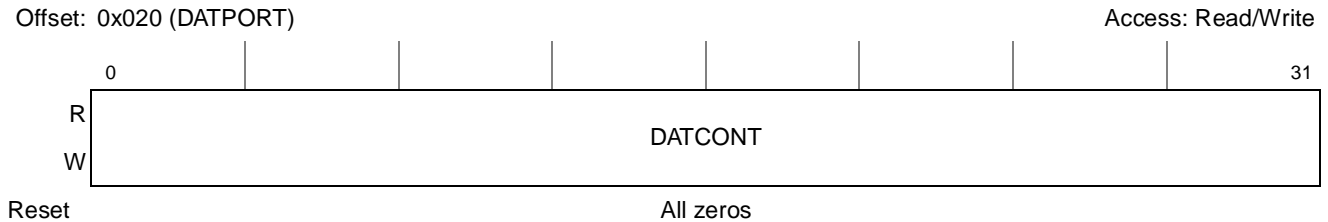


Figure 11-8. Buffer Data Port Register (DATPORT)

Table 11-10 describes the DATPORT fields.

Table 11-10. DATPORT Field Descriptions

Bit	Name	Description
0–31	DATCONT	Data content. The buffer data port register is for 32-bit data access by the CPU. When the internal DMA is enabled, any write to this register is ignored, and a read from this register always yields 0.

## 11.4.7 Present State Register (PRSSSTAT)

The present state register (PRSSSTAT), shown in Figure 11-9, indicates the status of the eSDHC to the host driver.

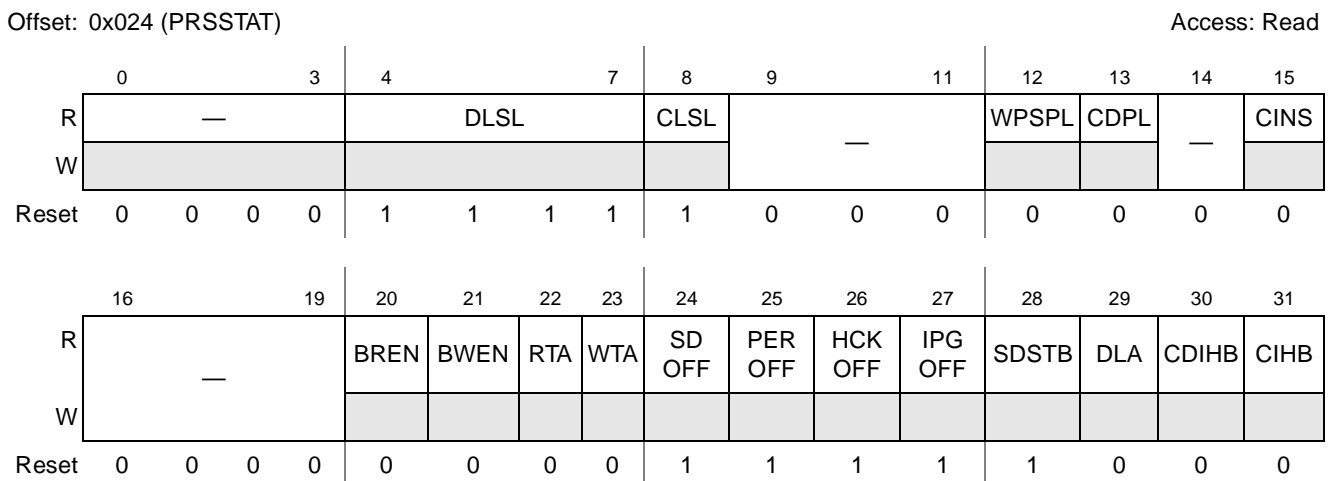


Figure 11-9. Present State Register (PRSSSTAT)

Table 11-11 describes the PRSSTAT fields.

**Table 11-11. PRSSTAT Field Descriptions**

Bit	Name	Description										
0–3	—	Reserved										
4–7	DLSL	<p>SD_DAT[3:0] line signal level. These bits are used to check the SD_DAT line level to recover from errors, and for debugging. This is especially useful in detecting the busy signal level from SD_DAT[0]. The reset value is affected by the external pull resistors. By default, read value of this bit field after reset is 0111, when SD_DAT[3] is pull-down and other lines are pull-up.</p> <table border="1" data-bbox="727 527 1110 774"> <thead> <tr> <th>PRSSTAT Bit</th> <th>SD_DAT<math>n</math></th> </tr> </thead> <tbody> <tr> <td>4</td> <td>3</td> </tr> <tr> <td>5</td> <td>2</td> </tr> <tr> <td>6</td> <td>1</td> </tr> <tr> <td>7</td> <td>0</td> </tr> </tbody> </table>	PRSSTAT Bit	SD_DAT $n$	4	3	5	2	6	1	7	0
PRSSTAT Bit	SD_DAT $n$											
4	3											
5	2											
6	1											
7	0											
8	CLSL	SD_CMD line signal level. This status is used to check the SD_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										
12	WPSPL	<p>Write protect switch pin level. The write protect switch is supported for memory and combo cards. This bit reflects the SD_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. If the SD_WP pin is not used, it should be tied to 0 so that the reset value of this bit is 0 and write is enabled.</p> <p>0 Write protected (SD_WP = 1) 1 Write enabled (SD_WP = 0)</p>										
13	CDPL	<p>Card detect pin level. This bit reflects the inverse value of the <math>\overline{\text{SD\_CD}}</math> 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. The reset value is affected by the external card detection pin. If this bit is not used, it should be tied to 0.</p> <p>0 No card present (<math>\overline{\text{SD\_CD}} = 1</math>) 1 Card present (<math>\overline{\text{SD\_CD}} = 0</math>)</p>										
14	—	Reserved										
15	CINS	<p>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. The software reset for all in the system control register does not affect this bit. A software reset does not affect this bit.</p> <p>0 Power-on-reset or no card 1 Card inserted</p>										
16–19	—	Reserved										

Table 11-11. PRSSTAT Field Descriptions (continued)

Bit	Name	Description
20	BREN	<p>Buffer read enable. This status is used for non-DMA read transfers. The eSDHC allows for multiple data buffers in the internal memory. This read-only flag, when set, indicates that valid data greater than watermark level exists in the host-side buffer.</p> <p>When the buffer is read, this bit is cleared. When valid data greater than watermark level is ready in the buffer, this bit is set and a buffer read ready interrupt is generated (if the interrupt is enabled).</p> <p>0 Buffer read disable 1 Buffer read enable</p>
21	BWEN	<p>Buffer write enable. This status is used for non-DMA write transfers. The eSDHC allows for multiple data buffers in the internal memory. This read-only flag, when set, indicates if space is available for greater than watermark level of write data.</p> <p>When the buffer is written, this bit is cleared. When the buffer can hold data greater than the watermark level, this bit is set and a buffer write ready interrupt is generated (if the interrupt is enabled).</p> <p>0 Buffer write disable 1 Buffer write enable</p>
22	RTA	<p>Read transfer active. This status is used for detecting completion of a read transfer. This bit is set for either of the following conditions:</p> <ul style="list-style-type: none"> <li>• After the end bit of the read command</li> <li>• When writing a 1 to PROCTL[CREQ] to restart a read transfer</li> </ul> <p>This bit is cleared for either of the following conditions:</p> <ul style="list-style-type: none"> <li>• When the last data block as specified by block length is transferred to the system</li> <li>• 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.</li> </ul> <p>0 No valid data 1 Transferring data</p>
23	WTA	<p>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.</p> <p>This bit is set in either of the following cases:</p> <ul style="list-style-type: none"> <li>• After the end bit of the write command.</li> <li>• When writing a 1 to PROCTL[CREQ] to restart a write transfer.</li> </ul> <p>This bit is cleared in either of the following cases:</p> <ul style="list-style-type: none"> <li>• After getting the CRC status of the last data block, as specified by the transfer count (single and multiple)</li> <li>• After getting the CRC status of any block where data transmission is about to be stopped by a stop-at-block-gap request.</li> </ul> <p>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.</p> <p>0 No valid data 1 Transferring data</p>
24	SDOFF	<p>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.</p> <p>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.</p>
25	PEROFF	<p>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.</p> <p>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.</p>

Table 11-11. PRSSTAT Field Descriptions (continued)

Bit	Name	Description
26	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. 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	IPGOFF	Controller clock gated off internally. Indicates that the controller clock is internally gated off. This bit is for the host driver to debug. The controller clock runs at <code>csb_clk / SCCR[ESDHCCM]</code> . 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	SDSTB	SD Clock Stable This status bit indicates that the internal card clock is stable. This bit is for the Host Driver to poll clock status when changing the clock frequency. It is recommended to clear <code>SDCLKEN</code> bit in System Control register to remove glitch on the card clock when the frequency is changing.  0 clock is changing frequency and not stable 1 clock is stable
29	DLA	Data line active. Indicates whether one of the <code>SD_DAT</code> line on SD bus is in use.  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. This bit is set in either of the following cases: <ul style="list-style-type: none"> <li>• After the end bit of the read command</li> <li>• When writing a 1 to <code>PROCTL[CREQ]</code> to restart a read transfer</li> </ul> This bit is cleared in either of the following cases: <ul style="list-style-type: none"> <li>• When the end bit of the last data block is sent from the SD bus to the eSDHC</li> <li>• When beginning a read wait transfer initiated by a stop at block gap request</li> </ul> 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.  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. This bit is set in any of the following cases: <ul style="list-style-type: none"> <li>• After the end bit of the write command</li> <li>• When writing a 1 to <code>PROCTL[CREQ]</code> to continue a write transfer</li> </ul> This bit is cleared in any of the following cases: <ul style="list-style-type: none"> <li>• 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.</li> <li>• When the SD card releases write-busy prior to waiting for write transfer as a result of a stop at block gap request</li> </ul> 0 <code>SD_DAT</code> line inactive 1 <code>SD_DAT</code> line active



Table 11-11. PRSSTAT Field Descriptions (continued)

Bit	Name	Description
30	CDIHB	Command inhibit (SD_DAT). This bit is set if the SD_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 (SD_DAT) (e.g. R1b and R5b type). Clearing from 1 to 0 generates a transfer complete interrupt. <b>Note:</b> The SD host driver can save registers for a suspend transaction after this bit has cleared from 1 to 0. 0 Can issue command which uses the SD_DAT line 1 Cannot issue command which uses the SD_DAT line
31	CIHB	Command inhibit (SD_CMD). This bit is cleared, if the SD_CMD line is not in use and the eSDHC can issue a SD/MMC command using the SD_CMD line. 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 SD_CMD line can be issued if this bit is cleared. Clearing from 1 to 0 generates a command complete interrupt. 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. 0 Can issue command using only SD_CMD line 1 Cannot issue command

**NOTE**

The host driver can issue CMD0, CMD12, CMD13 (for memory) and CMD52 (for SDIO) when the SD\_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.

**11.4.8 Protocol Control Register (PROCTL)**

The protocol control register is shown in Figure 11-10.

Offset: 0x028 (PROCTL)

Access: Read/Write

	0	4	5	6	7	8	11	12	13	14	15				
R	—				WE CRM	WE CINS	WE CINT	—				IABG	RW CTL	CREQ	SABG REQ
W	—				WE CRM	WE CINS	WE CINT	—				IABG	RW CTL	CREQ	SABG REQ
Reset	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	—				CDSS	CDTL	EMODE		D3CD	DTW		—			
W	—				CDSS	CDTL	EMODE		D3CD	DTW		—			
Reset	0	0	0	0	0	0	0	1	0	0	0	0			

Figure 11-10. Protocol Control Register (PROCTL)

Table 11-12 describes the PROCTL fields.

**Table 11-12. PROCTL Field Descriptions**

Bit	Name	Description
0–4	—	Reserved
5	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. 0 Disable 1 Enable
6	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. 0 Disable 1 Enable
7	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. 0 Disable 1 Enable
8–11	—	Reserved
12	IABG	Interrupt at block gap. This bit is valid only in 4-bit mode of the SDIO card and selects a sample point in the interrupt cycle. If the SDIO card cannot signal an interrupt during a multiple block transfer, this bit should be cleared to avoid an inadvertent interrupt. When the host driver detects an SDIO card insertion, it should set this bit according to the CCCR of the card. 0 Disable interrupt detection during a multiple block transfer. 1 Enable interrupt detection at the block gap for a multiple block transfer.
13	RWCTL	Read wait control. The read wait function is optional for SDIO cards. If the card supports read wait, set this bit to enable the read wait protocol to stop read data using the SD_DAT[2] line. Otherwise, the eSDHC has to stop the SD clock to hold read data, which restricts command generation. When the host driver detects an SDIO card insertion, it should set this bit according to the CCCR of the card. If the card does not support read wait, this bit should never be set otherwise an SD_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. 0 Disable read-wait control, and stop SD clock at block gap when the SABGREQ bit is set 1 Enable read-wait control, and assert read wait without stopping the SD clock at block gap when PROCTL[SABGREQ] is set
14	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. The eSDHC automatically clears this bit in either of the following cases: <ul style="list-style-type: none"> <li>• For a read transaction, the PRSSTAT[DLA] bit changes from 0 to 1 as a read transaction restarts.</li> <li>• For a write transaction, the PRSSTAT[WTA] bit changes from 0 to 1 as the write transaction restarts.</li> </ul> Therefore, it is not necessary for the host driver to clear. If SABGREQ is set, writes to CREQ would be ignored. 0 No effect 1 Restart

Table 11-12. PROCTL Field Descriptions (continued)

Bit	Name	Description
15	SABGREQ	<p>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. Read wait is used to stop the read transaction at the block gap. The eSDHC honors stop-at-block-gap request for write transfers. But for read transfers it requires that the SDIO card support read wait. Therefore, the host driver should not set this bit during read transfers unless the SDIO card supports read wait and has set read wait control to 1. Otherwise, the eSDHC stops the SD bus clock to pause the read operation during the block gap.</p> <p>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.</p> <p>This bit affects PRSSTAT[RTA, WTA, DLA, CIHB].</p> <p>0 Transfer 1 Stop or not resume yet</p>
16–23	—	Reserved
24	CDSS	<p>Card detect signal selection. Selects the source for card detection.</p> <p>If CDSS = 0 and D3CD = 0, then <math>\overline{SD\_CD}</math> is used.</p> <p>If CDSS = 0 and D3CD = 1 then DAT3 will be used for card detect</p> <p>0 Card detection level is selected (for normal purpose) 1 Card detection test level is selected (for test purpose)</p>
25	CDTL	<p>Card detect test level. Determines card insertion status when CDSS is set.</p> <p>0 No card in the slot 1 Card is inserted</p>
26–27	EMODE	<p>Endian mode. eSDHC supports only address-invariant mode in data transfer.</p> <p>00 Reserved 01 Reserved 10 Address-invariant mode. Each byte location in the main memory is mapped to the same byte location in the MMC/SD card. 11 Reserved</p>
28	D3CD	<p>SD_DAT3 as card detection pin. If this bit is set, SD_DAT3 should be pulled down to act as a card detection pin. Be cautious when using this feature, because SD_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.</p> <p>0 SD_DAT3 does not monitor card insertion 1 SD_DAT3 is card detection pin</p>
29–30	DTW	<p>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.</p> <p>00 1-bit mode 01 4-bit mode 10 Reserved 11 Reserved</p>
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.

### 11.4.9 System Control Register (SYSCTL)

The system control register is shown in [Figure 11-11](#).

Offset: 0x02C (SYSCTL)

Access: Mixed

	0	3	4	5	6	7	8	11	12	15		
R	—			INITA			—			DTCV		
W	—				RSTD	RSTC	RSTA	—			DTCV	
Reset	0	0	0	0	0	0	0	0	0	0	0	
	16	23	24	27	28	29	30	31				
R	SDCLKFS				DVS			CLKE N	PEREN	HCKEN	IPGEN	
W	SDCLKFS				DVS			CLKE N	PEREN	HCKEN	IPGEN	
Reset	1	0	0	0	0	0	0	0	1	0	0	

**Figure 11-11. System Control Register (SYSCTL)**

[Table 11-13](#) describes the SYSCTL fields.

**Table 11-13. SYSCTL Field Descriptions**

Bit	Name	Description
0–3	—	Reserved
4	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. 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.

Table 11-13. SYSCTL Field Descriptions (continued)

Bit	Name	Description
5	RSTD	Software reset for SD_DAT line. The DMA and part of the data circuit are reset. The following registers and bits are cleared by this bit: <ul style="list-style-type: none"> <li>• DATPORT register</li> <li>• Buffer is cleared and initialized; PRSSTAT register</li> <li>• PRSSTAT[BREN, BWEN, RTA, WTA, DLA, CDIHB]</li> <li>• PROCTL[CREQ, SABGREQ]</li> <li>• IRQSTAT[BRR, BWR, DINT, BGE, TC]</li> </ul> 0 Work 1 Reset
6	RSTC	Software reset for SD_CMD line. Only part of the command circuit is reset. The following bits are cleared by this bit: <ul style="list-style-type: none"> <li>• PRSSTAT[CIHB]</li> <li>• IRQSTAT[CC]</li> </ul> 0 Work 1 Reset
7	RSTA	Software reset for all. This reset affects the entire host controller except for the card-detection circuit. Register bits of type Read only, Read/Write, Read only: write-1-to-clear, and Read/Write Automatic clear are cleared. 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 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. 0 Work 1 Reset
8–11	—	Reserved
12–15	DTOCV	Data timeout counter value. Determines the interval by which SD_DAT line timeouts are detected. Refer to the data timeout error <a href="#">Section 11.4.10, “Interrupt Status Register (IRQSTAT)”</a> , for information on factors that dictate timeout generation. Timeout clock frequency is generated by dividing the base clock SD_CLK value by this value. When setting this register, prevent inadvertent timeout events by clearing IRQSTATEN[DTOESEN]. 0000 SD_CLK x 2 <sup>13</sup> 0001 SD_CLK x 2 <sup>14</sup> ... 1110 SD_CLK x 2 <sup>27</sup> 1111 Reserved

Table 11-13. SYSTL Field Descriptions (continued)

Bit	Name	Description
16–23	SDCLKFS	<p>SD_CLK frequency select. This field, together with DVS, selects the frequency of SD_CLK pin. This bit holds the prescaler of the base clock frequency. Only the following settings are allowed:</p> <p>0x01 Base clock divided by 2  0x02 Base clock divided by 4  0x04 Base clock divided by 8  0x08 Base clock divided by 16  0x10 Base clock divided by 32  0x20 Base clock divided by 64  0x40 Base clock divided by 128  0x80 Base clock divided by 256</p> <p>Multiple bits must not be set or the behavior of this prescaler is undefined.  According to the <i>SD Physical Specification</i> and the <i>SDIO Card Specification version 1.2 2.0</i>, the maximum SD clock frequency is 50 MHz, and should never exceed this limit. The frequency of SD_CLK is set by the following formula:</p> $\text{clock frequency} = (\text{base clock}) / [(\text{SDCLKFS} \times 2) \times (\text{DVS} + 1)] \quad \text{Eqn. 11-1}$ <p>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. 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.</p> <p><b>Note:</b> The base clock frequency equals the <code>csb_clk / SCCR[SDHCCM]</code>.</p>
24–27	DVS	<p>Divisor. Provides a more exact divisor to generate the desired SD clock frequency. The settings are as follows:</p> <p>0x0 Divide by 1  0x1 Divide by 2  ...  0xE Divide by 15  0xF Divide by 16</p>
28	CLKEN	<p>SD Card Clock Enable</p> <p>0 Disable the clock  1 Enable the clock</p>

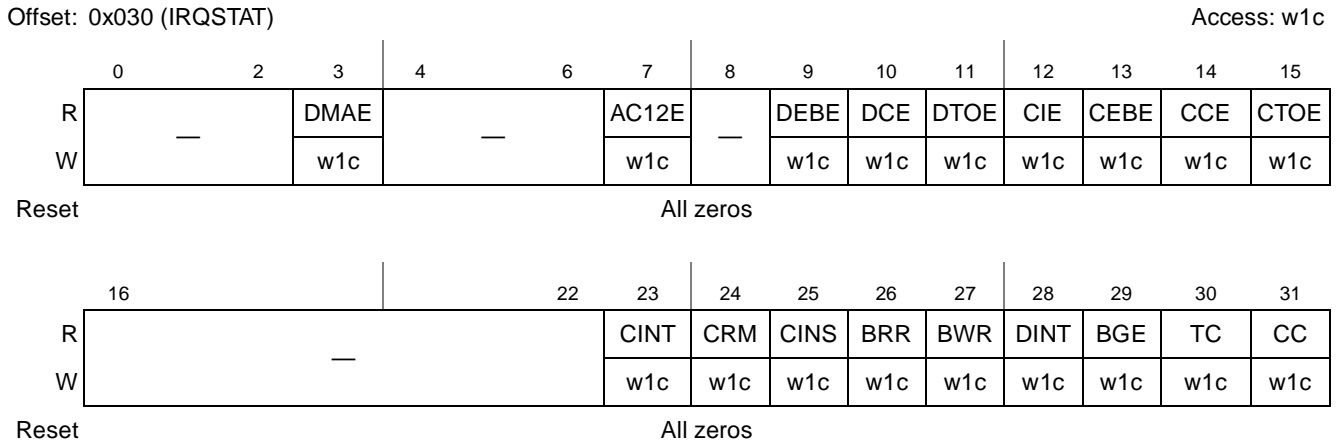
Table 11-13. SYSTL Field Descriptions (continued)

Bit	Name	Description
29	PEREN	<p>Peripheral clock enable. If set, the peripheral clock is always active and no automatic gating is applied, thus SD_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 SD_CLK immediately. The peripheral clock will be internally gated off, if none of the following factors are met:</p> <ul style="list-style-type: none"> <li>• Command part is reset</li> <li>• Data part is reset</li> <li>• Soft reset</li> <li>• Command is about to send</li> <li>• Clock divisor is just updated</li> <li>• Continue request is just set</li> <li>• This bit is set</li> <li>• Card insertion is detected</li> <li>• Card removal is detected</li> <li>• Card external interrupt is detected</li> <li>• 80 clocks for initialization phase is ongoing</li> </ul> <p>0 The peripheral clock is internally gated off 1 The peripheral clock is not automatically gated off</p>
30	HCKEN	<p>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.</p> <p><b>Note:</b> Master clock is the clock to the DMA engine and to the CSB interface logic.</p> <p>0) Master clock is internally gated off 1) Master clock is not automatically gated off</p>
31	IPGEN	<p>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:</p> <ul style="list-style-type: none"> <li>• Command part is reset</li> <li>• Data part is reset</li> <li>• Soft reset</li> <li>• Command is about to send</li> <li>• Clock divisor is just updated</li> <li>• Continue request is just set</li> <li>• This bit is set</li> <li>• Card insertion is detected</li> <li>• Card removal is detected</li> <li>• Card external interrupt is detected</li> <li>• The controller clock is not gated off</li> </ul> <p><b>Note:</b> 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 SYSTL[PEREN] is not cleared.</p> <p>0 The controller clock is internally gated off 1 The controller clock is not automatically gated off</p>

### 11.4.10 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.

Figure 11-12 shows the interrupt status register.



**Figure 11-12. Interrupt Status Register (IRQSTAT)**

Table 11-14 describes the IRQSTAT fields.

**Table 11-14. IRQSTAT Field Descriptions**

Bit	Name	Description
0–2	—	Reserved
3	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. 0 No Error 1 Error
4–6	—	Reserved
7	AC12E	Auto CMD12 error. Occurs when one of the bits in AUTO12ERR is set. This bit is also set when Auto CMD12 is not executed due to a previous command error. 0 No Error 1 Error
8	—	Reserved
9	DEBE	Data end bit error. Occurs when detecting 0 at the end bit position of read data on the SD_DAT line or at the end bit position of the CRC. 0 No Error 1 Error <b>Note:</b> 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	DCE	Data CRC error. Occurs when detecting CRC error when transferring read data on the SD_DAT line or when detecting the write CRC status having a value other than 0b010. 0 No Error 1 Error



Table 11-14. IRQSTAT Field Descriptions (continued)

Bit	Name	Description
11	DTOE	Data timeout error. Occurs during one of following timeout conditions: <ul style="list-style-type: none"> <li>• Busy timeout for R1b and R5b types</li> <li>• Busy timeout after write CRC status</li> <li>• Read data timeout</li> </ul> 0 No error 1 Timeout
12	CIE	Command index error. Occurs if a command index error occurs in the command response. 0 No error 1 Timeout [Error]
13	CEBE	Command end bit error. Occurs when the end bit of a command response is 0. 0 No error 1 End bit error generated
14	CCE	Command CRC error. A command CRC error is generated in two cases: <ul style="list-style-type: none"> <li>• 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.</li> <li>• The eSDHC detects a SD_CMD line conflict by monitoring the SD_CMD line when a command is issued. If the eSDHC drives the SD_CMD line to 1, but detects 0 on the SD_CMD line at the next SD_CLK edge, then the eSDHC aborts the command (stop driving SD_CMD line) and sets this bit. The CTOE bit is also set to distinguish the SD_CMD line conflict.</li> </ul> 0 No error 1 CRC error generated
15	CTOE	Command timeout error. Occurs if no response is returned within 64 SD_CLK cycles from the end bit of the command. Also, if eSDHC detects a SD_CMD line conflict, this bit is set along with IRQSTAT[CCE] as shown in <a href="#">Table 11-26</a> . 0 No error 1 Time out
16–22	—	Reserved
23	CINT	Card interrupt. <ul style="list-style-type: none"> <li>• In 1-bit mode, the eSDHC detects the card interrupt without the SD clock to support wakeup.</li> <li>• 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. 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.</li> </ul> <p>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.</p> 0 No card interrupt 1 Generate card interrupt
24	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. When this bit is cleared, it is set again if no card is inserted. To leave it cleared, clear IRQSTATEN[CRMSEN]. 0 Card state unstable or inserted 1 Card removed

Table 11-14. IRQSTAT Field Descriptions (continued)

Bit	Name	Description
25	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. When this bit is cleared, it is set again if a card has been inserted. To leave it cleared, clear IRQSTATEN[CINSEN]. 0 Card state unstable or removed 1 Card inserted
26	BRR	Buffer read ready. This bit is set if PRSSTAT[BREN] changes from 0 to 1. 0 Not ready to read buffer 1 Ready to read buffer
27	BWR	Buffer write ready. This bit is set if PRSSTAT[BWEN] changes from 0 to 1. 0 Not ready to write buffer 1 Ready to write buffer
28	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. 0 No DMA interrupt 1 DMA interrupt is generated
29	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. During a read transaction, this bit is set at the falling edge of the SD_DAT line active status (when the transaction is stopped at SD bus timing). Read wait must be supported to use this function. During a write transaction, this bit is set at the falling edge of PRSSTAT[WTA] (after reading the CRC status at SD bus timing). 0 No block gap event 1 Transaction stopped at block gap
30	TC	Transfer complete. This bit is set when a read or write transfer is completed. 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: <ul style="list-style-type: none"> <li>When a data transfer is completed, as specified by data length (after the last data has been read to the host system).</li> <li>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).</li> </ul> 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: <ul style="list-style-type: none"> <li>When the last data is written to the SD card, as specified by data length and the busy signal is released.</li> <li>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).</li> </ul>
31	CC	Command complete. This bit is set when the end bit of the command response is received (except Auto CMD12). Refer to PRSSTAT[CIHB]. 0 No command complete 1 Command complete

Table 11-15 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 11-15. 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 SD_CLK cycles
1	0	Response received

Table 11-16 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 11-16. 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 11-17 below.

**Table 11-17. 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	SD_CMD line conflict

### 11.4.11 Interrupt Status Enable Register (IRQSTATEN)

Figure 11-13 shows the interrupt status enable register. 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

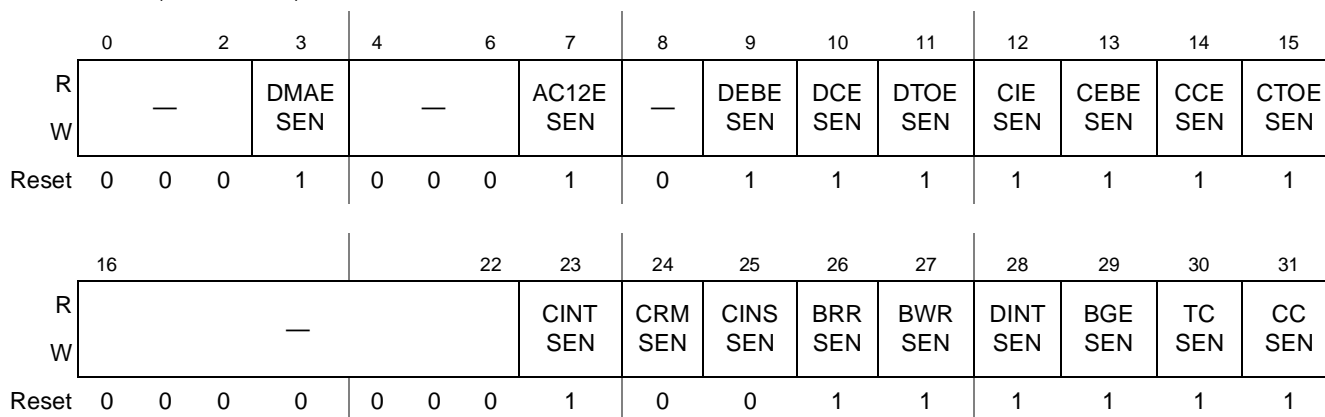


Figure 11-13. Interrupt Status Enable Register (IRQSTATEN)

Table 11-18 describes the IRQSTATEN fields.

Table 11-18. IRQSTATEN Field Descriptions

Bit	Name	Description
0–2	—	Reserved
3	DMAESEN	DMA error status enable 0 Masked 1 Enabled
4–6	—	Reserved
7	AC12ESEN	Auto CMD12 error status enable 0 Masked 1 Enabled
8	—	Reserved
9	DEBESEN	Data end bit error status enable 0 Masked 1 Enabled
10	DCESEN	Data CRC error status enable 0 Masked 1 Enabled
11	DTOESEN	Data timeout error status enable 0 Masked 1 Enabled
12	CIESEN	Command index error status enable 0 Masked 1 Enabled
13	CEBESEN	Command end bit error status enable 0 Masked 1 Enabled
14	CCesen	Command CRC error status enable 0 Masked 1 Enabled

Table 11-18. IRQSTATEN Field Descriptions (continued)

Bit	Name	Description
15	CTOESEN	Command timeout error status enable 0 Masked 1 Enabled
16–22	—	Reserved
23	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. 0 Masked 1 Enabled
24	CRMSEN	Card removal status enable 0 Masked 1 Enabled
25	CINSEN	Card insertion status enable 0 Masked 1 Enabled
26	BRRSEN	Buffer read ready status enable 0 Masked 1 Enabled
27	BWRSEN	Buffer write ready status enable 0 Masked 1 Enabled
28	DINTSEN	DMA interrupt status enable 0 Masked 1 Enabled
29	BGESEN	Block gap event status enable 0 Masked 1 Enabled
30	TCSSEN	Transfer complete status enable 0 Masked 1 Enabled
31	CCSEN	Command complete status enable 0 Masked 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 SD\_CMD line conflict, the host driver must set both CTOESEN and CCSEN bits.

## 11.4.12 Interrupt Signal Enable Register (IRQSIGEN)

IRQSIGEN, shown in Figure 11-14, 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

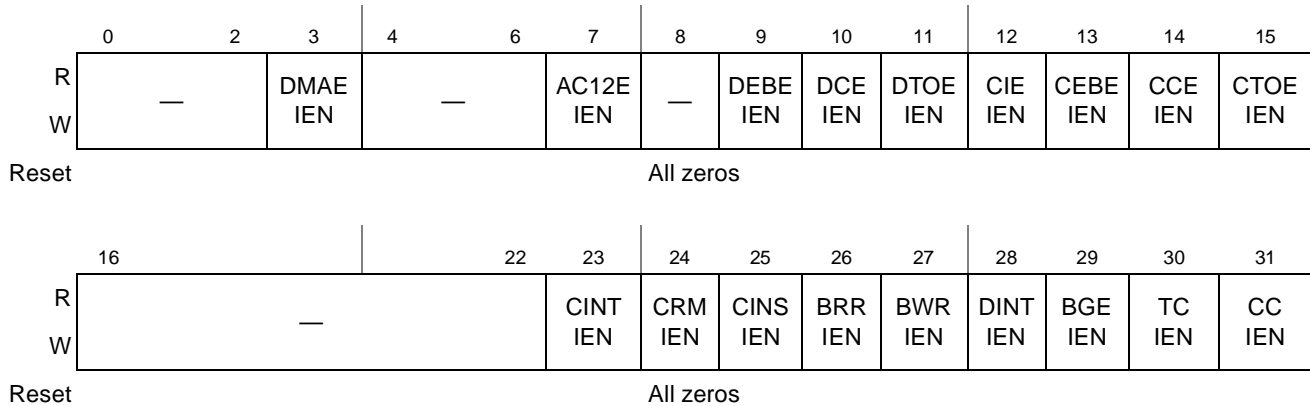


Figure 11-14. Interrupt Signal Enable Register (IRQSIGEN)

Table 11-19 describes the IRQSIGEN fields.

Table 11-19. IRQSIGEN Field Descriptions

Bit	Name	Description
0–2	—	Reserved
3	DMAEIEN	DMA error interrupt enable 0 Masked 1 Enabled
4–6	—	Reserved
7	AC12EIEN	Auto CMD12 error interrupt enable 0 Masked 1 Enabled
8	—	Reserved
9	DEBEIEN	Data end bit error interrupt enable 0 Masked 1 Enabled
10	DCEIEN	Data CRC error interrupt enable 0 Masked 1 Enabled
11	DTOEIEN	Data timeout error interrupt enable 0 Masked 1 Enabled

Table 11-19. IRQSIGEN Field Descriptions (continued)

Bit	Name	Description
12	CIEIEN	Command index error interrupt enable 0 Masked 1 Enabled
13	CEBEIEN	Command end bit error interrupt enable 0 Masked 1 Enabled
14	CCEIEN	Command CRC error interrupt enable 0 Masked 1 Enabled
15	CTOEIEN	Command timeout error interrupt enable 0 Masked 1 Enabled
16–22	—	Reserved
23	CINTIEN	Card interrupt signal enable 0 Masked 1 Enabled
24	CRMIEN	Card removal interrupt enable 0 Masked 1 Enabled
25	CINSIEN	Card insertion interrupt enable 0 Masked 1 Enabled
26	BRIEN	Buffer read ready interrupt enable 0 Masked 1 Enabled
27	BWRIEN	Buffer write ready interrupt enable 0 Masked 1 Enabled
28	DINTIEN	DMA interrupt enable 0 Masked 1 Enabled
29	BGEIEN	Block gap event interrupt enable 0 Masked 1 Enabled
30	TCIEN	Transfer complete interrupt enable 0 Masked 1 Enabled
31	CCIEN	Command complete interrupt enable 0 Masked 1 Enabled

### 11.4.13 Auto CMD12 Error Status Register (AUTO12ERR)

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.

Figure 11-15 shows the auto CMD12 error status register.

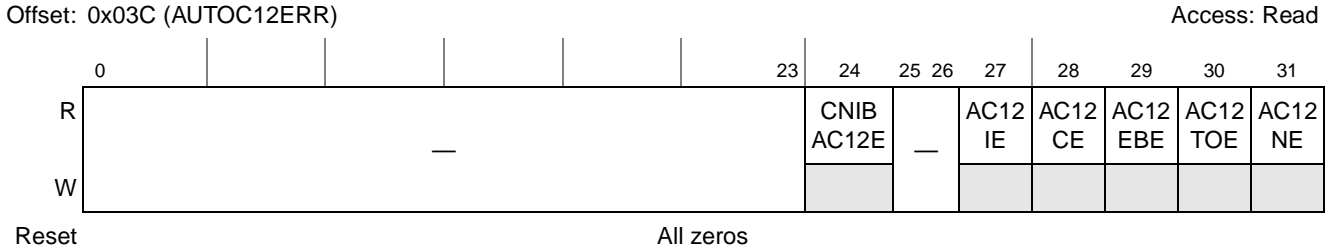


Figure 11-15. Auto CMD12 Error Status Register (AUTO12ERR)

Table 11-20 describes the IRQSTATEN fields.

Table 11-20. AUTO12ERR Field Descriptions

Bit	Name	Description
0–23	—	Reserved
24	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 (D27–D30). 0 No error 1 Not Issued
25–26	—	Reserved
27	AC12IE	Auto CMD12 index error. Occurs if the command index error occurs in response to a command. 0 No error 1 Error, the CMD index in response is not CMD12
28	AC12CE	Auto CMD12 CRC error. Occurs when detecting a CRC error in the command response. 0 No CRC error 1 CRC error met in Auto CMD12 response
29	AC12EBE	Auto CMD12 end bit error. Occurs when detecting that the end bit of command response is 0 when it should be 1. 0 No error 1 End bit error generated



**Table 11-20. AUTO12ERR Field Descriptions (continued)**

Bit	Name	Description
30	AC12TOE	Auto CMD12 timeout error. Occurs if no response is returned within 64 SD_CLK cycles from the end bit of the command. If this bit is set, the other error status bits (29–27) are meaningless. 0 No error 1 Time out
31	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 (30–27) are meaningless. 0 Executed 1 Not executed

Table 11-21 describes the relationship between command CRC error and command timeout error for Auto CMD12.

**Table 11-21. 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	SD_CMD line conflict

There are three scenarios when AUTO12ERR can be changed:

- 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.
- At the end bit of an Auto CMD12 response
  - Check received responses by checking the error bits 30–27.
  - Set if error is detected.
  - Clear if error is not detected.
- Before reading AUTO12ERR[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 31–27 is set to 1. The CNIBAC12E error bit does not generate an interrupt.

### 11.4.14 Host Controller Capabilities (HOSTCAPBLT)

The host controller capabilities provide the host driver with information specific to the eSDHC implementation. The value in this register does not change in software reset, and any write to this register is ignored.

Figure 11-16 shows the auto CMD12 error status register.

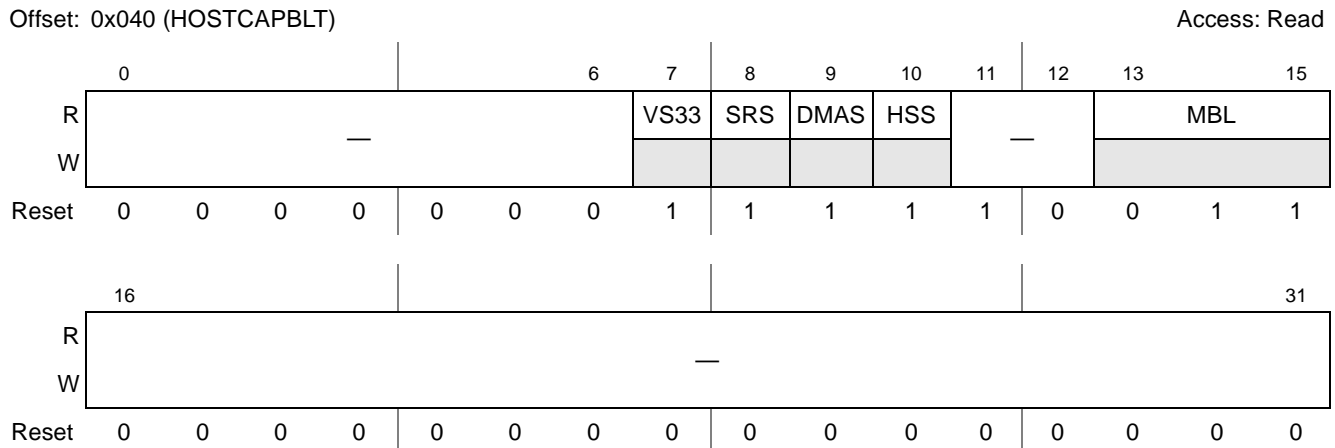


Figure 11-16. Host Capabilities Register (HOSTCAPBLT)

Table 11-22 describes the HOSTCAPBLT fields.

Table 11-22. HOSTCAPBLT Field Descriptions

Bit	Name	Description
0–6	—	Reserved
7	VS33	Voltage support 3.3 V. This bit depends on the host system ability. 0 3.3 V not supported 1 3.3 V supported Note: This is always set to 1.
8	SRS	Suspend/resume support. Indicates if eSDHC supports suspend/resume functionality. If this bit is 0, suspend and resume mechanism, as well as the read wait, are not supported and the host driver should not issue suspend or resume commands. 0 Not supported 1 Supported
9	DMAS	DMA support. Indicates if eSDHC is capable of using internal DMA to transfer data between system memory and the data buffer directly. 0 DMA not supported 1 DMA supported
10	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. 0 High speed supported 1 High speed supported
11–12	—	Reserved. Set to 10.

Table 11-22. HOSTCAPBLT Field Descriptions (continued)

Bit	Name	Description
13–15	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. 000 512 bytes 001 1024 bytes 010 2048 bytes 011 4096 bytes
16–31	—	Reserved

### 11.4.15 Watermark Level Register (WML)

Figure 11-17 shows the watermark level register. Both write and read watermark levels are configurable. The value can be any number from 1–127 words.

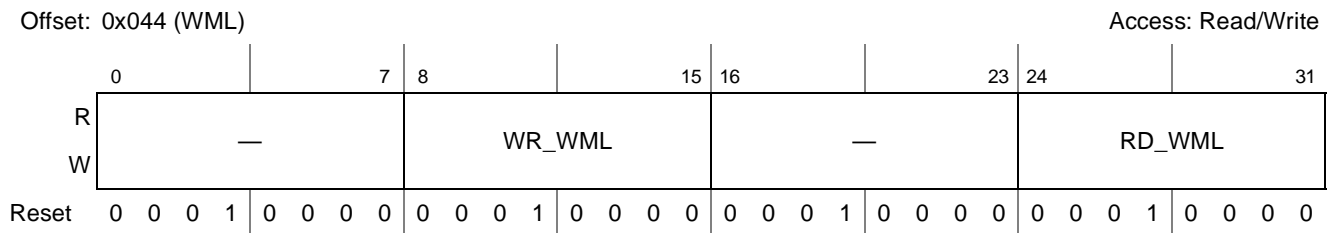


Figure 11-17. Watermark Level Register (WML)

Table 11-23 describes the WML fields.

Table 11-23. WML Field Descriptions

Bit	Name	Description
0–7	—	Reserved.
8–15	WR_WML	Write watermark level. Number of 32-bit words of watermark level in write data transfer. <b>Note:</b> The minimum value is 0x02, which represents 2 words (8 bytes).
16–23	—	Reserved.
24–31	RD_WML	Read watermark level. Number of 32-bit words of watermark level in read data transfer.

### 11.4.16 Force Event Register (FEVT)

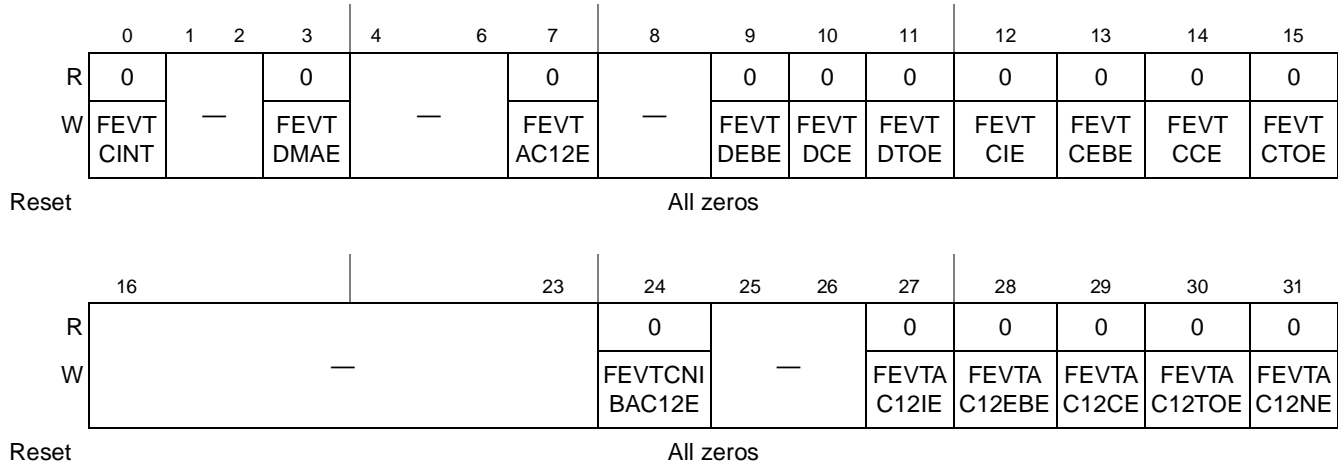
The force event register is not a physically implemented register. Rather, it is an address to which the IRQSTAT register can be written if the corresponding bit of IRQSTATEN is set. Therefore, this register is a write-only register and writing zero has no effect. Writing 1 to this register sets the corresponding bit of IRQSTAT. Reading from this register always returns zeroes.

Forcing a card interrupt generates a short pulse on the SD\_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.

Figure 11-18 shows the force event register.

Offset: 0x050 (FEVT)

Access: Write



**Figure 11-18. Force Event Register (FEVT)**

Table 11-24 describes the FEVT fields.

**Table 11-24. FEVT Field Descriptions**

Bit	Name	Description
0	FEVTCINT	Force event card interrupt. Writing 1 to this bit generates a low-level short pulse on the internal SD_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	FEVTDMAE	Force event DMA error. Forces IRQSTAT[DMAE] to set.
4–6	—	Reserved
7	FEVTAC12E	Force event Auto CMD12 error. Forces IRQSTAT[AC12E] to set.
8	—	Reserved
9	FEVTDEBE	Force event data end bit error. Forces IRQSTAT[DEBE] to set.
10	FEVTDCE	Force event data CRC error. Forces IRQSTAT[DCE] to set.
11	FEVTDTOE	Force event data time out error. Forces IRQSTAT[DTOE] to set.
12	FEVTCIE	Force event command index error. Forces IRQSTAT[CCE] to set.
13	FEVTCEBE	Force event command end bit error. Forces IRQSTAT[CEBE] to set.
14	FEVTCCE	Force event command CRC error. Forces IRQSTAT[CCE] to set.
15	FEVTCCE	Force event command time out error. Forces IRQSTAT[CTOE] to set.
16–23	—	Reserved
24	FEVTCNIBAC12E	Force event command not executed by Auto CMD12 error. Forces AUTOC12ERR[CNIBAC12E] to set.
25–26	—	Reserved

Table 11-24. FEVT Field Descriptions (continued)

Bit	Name	Description
27	FEVTAC12IE	Force event Auto CMD12 index error. Forces AUTOC12ERR[AC12IE] to set.
28	FEVTAC12EBE	Force event Auto CMD12 end bit error. Forces AUTOC12ERR[AC12EBE] to set.
29	FEVTAC12CE	Force event Auto CMD12 CRC error. Forces AUTOC12ERR[AC12CE] to set.
30	FEVTAC12TOE	Force event Auto CMD12 time out error. Forces AUTOC12ERR[AC12TOE] to set.
31	FEVTAC12NE	Force event Auto CMD12 not executed. Forces AUTOC12ERR[AC12NE] to set.

### 11.4.17 Host Controller Version Register (HOSTVER)

The host controller version register, shown in Figure 11-19, contains the version for the vendor and the host controller. All the bits are read-only.

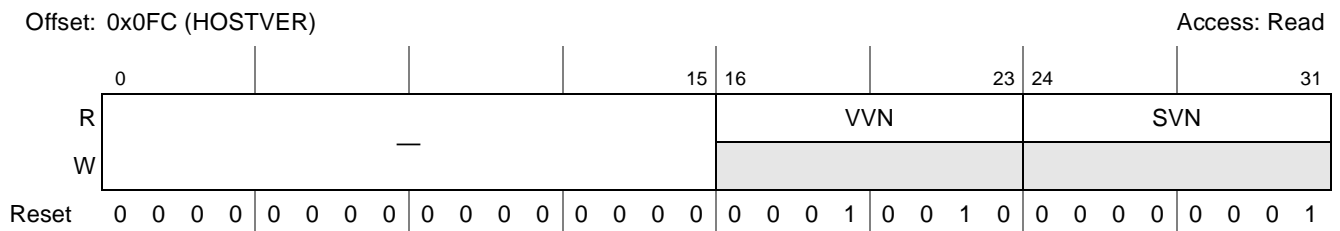


Figure 11-19. Host Controller Version Register (HOSTVER)

Table 11-25 describes the HOSTVER fields.

Table 11-25. HOSTVER Field Descriptions

Bit	Name	Description
0–15	—	Reserved
16–23	VVN	Vendor version number. The host driver should not use this status. The upper and the lower 4-bits indicate the version. 0x12 Freescale eSDHC <i>version 2.2</i> others Reserved
24–31	SVN	Specification version number. Indicates for the host controller specification version. 0x01 SD Host Specification <i>Version 2.0</i> , supports the test event register. others Reserved

### 11.4.18 DMA Control Register (DCR)

This is implemented as SDHCCR as described in Section 5.2.2.12.

## 11.5 Functional Description

The following sections provide a brief functional description of the major system blocks, including the data buffer, DMA CSB interface, register bank, register bus interface, dual-port memory wrapper, data/command controller, clock and reset manager, and clock generator.

## 11.5.1 Data Buffer

The eSDHC uses one configurable data buffer so that data can be transferred between the internal system bus (register bus or CSB 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 11-20 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 water mark levels for read and write are both configurable and can be any value between 1 and 127 words.

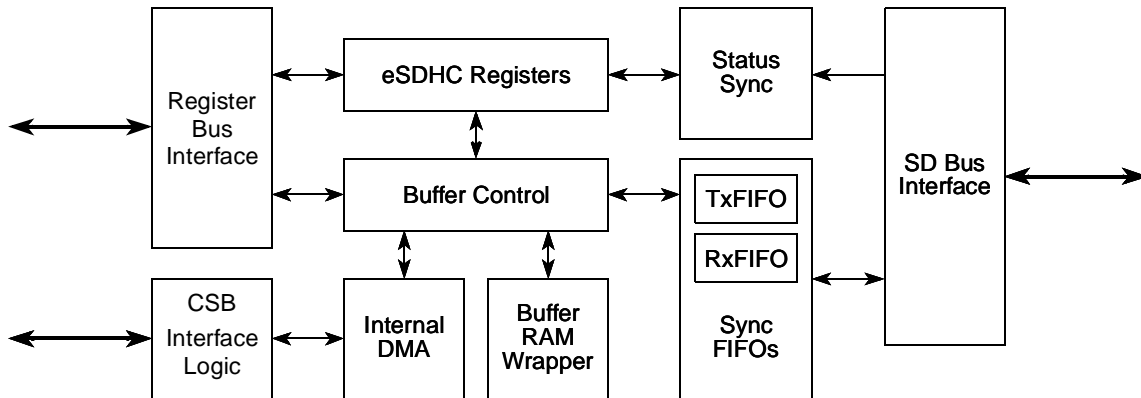


Figure 11-20. eSDHC Buffer Scheme

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 CSB 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 CSB access permission, the internal DMA burst-writes WR\_WML number of words into the buffer

### 11.5.1.1 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), and more than WML[WR\_WML] number of empty word slots are available and ready for receiving new data, the eSDHC sets the 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 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 write operation from that address. It is recommended that software reset for data is applied before the transfer is restarted after error recovery.

The eSDHC does not start fetching the data from the host system, 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 SD\_CLK to avoid a data buffer underrun situation.

### 11.5.1.2 Read Operation Sequence

There are two ways to read data from the buffer when transferring data from 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), and more than WML[RD\_WML] number of words are available and ready for the system to fetch data, the eSDHC sets the IRQSTAT[BRR]. 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 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 SD\_CLK to avoid a data buffer overrun situation.

### 11.5.1.3 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, SDIO, 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.

#### 11.5.1.4 Dividing Large Data Transfer

This SDIO command CMD53 definition limits the maximum data size of data transfers according to the following formula:

$$\text{Maximum data size} = (\text{block size}) \times (\text{block count}) \quad \text{Eqn. 11-2}$$

The length of a multiple block transfer must be in block size units. If the total data length cannot be divided evenly to a multiple of the block size, then there are two ways to transfer the data depending on the function and card design.

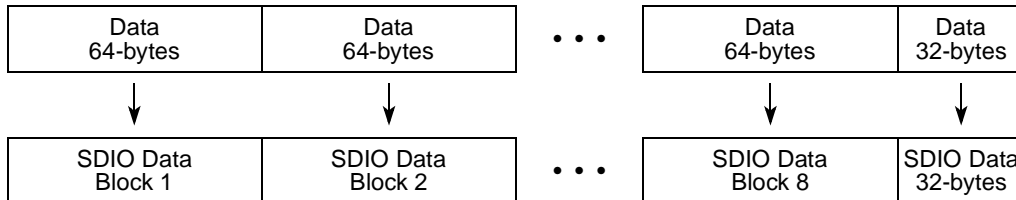
- The card driver splits the transaction. The remainder of block size data is then transferred using a single block command at the end.
- Add dummy data in the last block to fill the block size. The card must remove the dummy data.

See [Figure 11-21](#) for an example of dividing large data transfers. Although the eSDHC supports a block size of up to 4096 bytes, the example below illustrates a maximum of 64 bytes where the data must be divided.

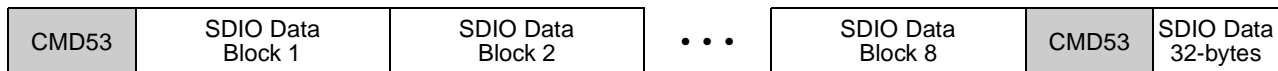
544-bytes WLAN Frame



WLAN Frame is divided equally into 64-byte blocks plus the remainder 32-bytes



Eight 64-byte blocks are sent in Block Transfer Mode and the remainder 32-bytes are sent in Byte Transfer Mode



**Figure 11-21. Example of Dividing a Large Data Transfer**

#### 11.5.2 DMA CSB Interface

The internal DMA implements a DMA engine and CSB 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 11-22](#) for illustration of the DMA CSB 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 through the DATPORT register.



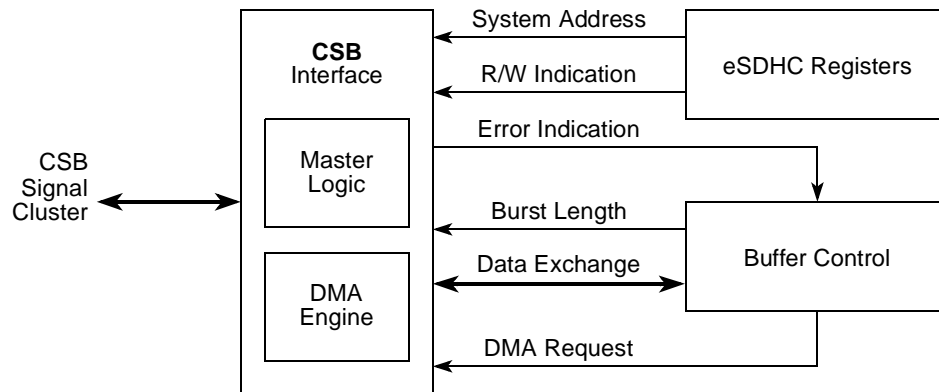


Figure 11-22. DMA CSB Interface Block

### 11.5.2.1 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 DMA engine. 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.

### 11.5.2.2 DMA Burst Length

DMA burst length is set to a default value to transfer the data into and out of the system bus. This value is not programmable by the user.

### 11.5.2.3 CSB 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.

## 11.5.3 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 SD\_DAT[0] line
- Monitors interrupt from the SDIO card
- 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.

### 11.5.3.1 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.

### 11.5.3.2 SD Clock and Monitor

This module monitors the signal level on all four 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  $\overline{\text{SD\_CD}}$  state, or signal level on SD\_DAT[3] line when PROCTL[D3CD] is set.

The module detects the SD\_WP (write protect) line. With the information of SD\_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.

### 11.5.3.3 Command Agent

The command agent deals with the transactions on SD\_CMD line. See [Figure 11-23](#) for illustration of the structure for the command CRC shift register.

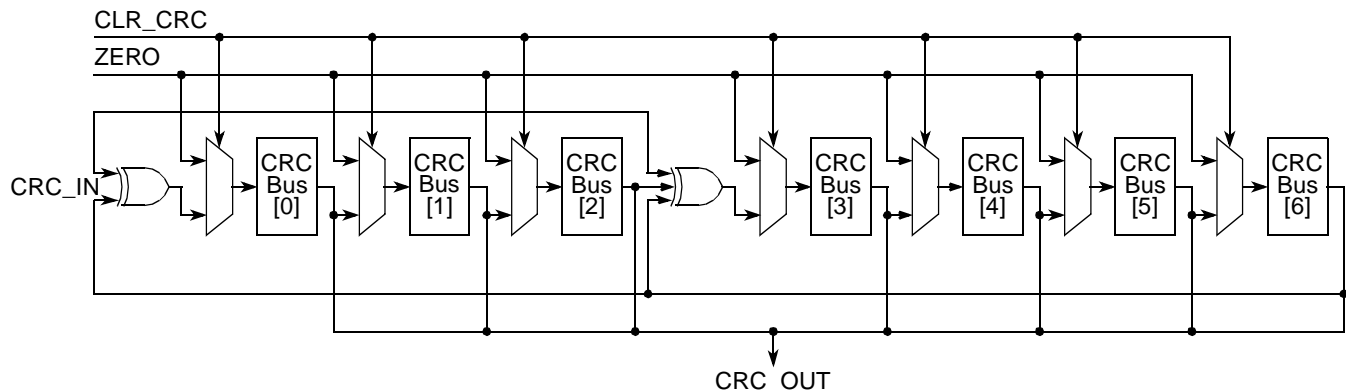


Figure 11-23. Command CRC Shift Register

The CRC polynomials for the SD\_CMD are as follows:

Generator polynomial:  $G(x) = x^7 + x^3 + 1$

$M(x) = (\text{first bit}) \times x^n + (\text{second bit}) \times x^{n-1} + \dots + (\text{last bit}) \times x^0$

$\text{CRC}[6:0] = \text{Remainder} [(M(x) \times x^7) \div G(x)]$

### 11.5.3.4 Data Agent

The data agent handles the transactions on the four data lines. Moreover, this module also detects the busy state from on SD\_DAT[0] line, and generates read wait state by the request from the transceiver. The CRC polynomials for the SD\_DAT are as follows:

Generator polynomial:  $G(x) = x^{16} + x^{12} + x^5 + 1$

$M(x) = (\text{first bit}) \times x^n + (\text{second bit}) \times x^{n-1} + \dots + (\text{last bit}) \times x^0$

$\text{CRC}[15:0] = \text{Remainder} [(M(x) \times x^{16}) \div G(x)]$

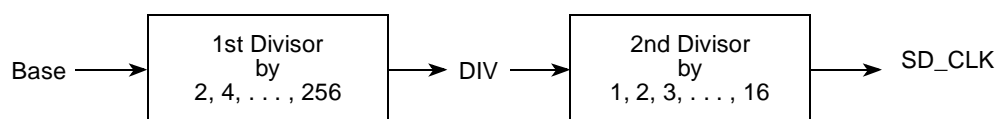
### 11.5.4 Clock & Reset Manager

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.

### 11.5.5 Clock Generator

The clock generator generates the SD\_CLK by dividing the internal bus clock into two stages. Refer to [Figure 11-24](#) for the structure of the divider, in which the term base represents the frequency of the internal bus clock. Refer to SYSCTL[SDCLKFS] and SYSCTL[DVS] (see [Section 11.4.9, “System Control Register \(SYSCTL\)”](#)) to select the divisor values.



**Figure 11-24. 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, SD\_CLK, as the driving clock for all sub-modules of SD protocol unit, and the sync FIFOs in [Figure 11-20](#) 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 SD\_CLK generated by the internal bus clock is base while the lowest frequency is base/4096.

### 11.5.6 SDIO Card Interrupt

This section discusses interrupts in 1- and 4-bit modes as well as card interrupt handling.

### 11.5.6.1 Interrupts in 1-bit Mode

In this case, the SD\_DAT[1] pin is dedicated to providing the interrupt function. An interrupt is asserted by pulling the SD\_DAT[1] low from the SDIO card, until the interrupt service is finished to clear the interrupt.

### 11.5.6.2 Interrupt in 4-bit Mode

As the interrupt and data line 1 share pin 8 in four-bit mode, an interrupt is only sent by the card and recognized by the host during a specific time. This is known as the interrupt period. The eSDHC only samples the level on pin 8 during the interrupt period. At all other times, the host ignores the level on pin 8 and treats it as the data signal. The definition of the interrupt period is different for operations with single- and multiple-block data transfers.

For normal single data block transmissions, the interrupt period becomes active two clock cycles after the completion of a data packet. This interrupt period lasts until after the card receives the end bit of the next command that has a data block transfer associated with it.

For multiple block data transfers in 4-bit mode there is only a limited period of time that the interrupt period can be active due to the limited period of data line availability between the multiple blocks of data. This requires a more strict definition of the interrupt period. For this case, the interrupt period is limited to two clock cycles, which begins two clocks after the end bit of the previous data block. During this two-clock cycle interrupt period if an interrupt is pending, the SD\_DAT[1] line is held low for one clock cycle with the last clock cycle pulling SD\_DAT[1] high. On completion of the interrupt period, the card releases the SD\_DAT[1] line into the high-Z state. The eSDHC samples the SD\_DAT[1] during the interrupt period when PROCTL[IABG] is set.

for further information about the SDIO card interrupt, see *SDIO Card Specification v2.0*.

### 11.5.6.3 Card Interrupt Handling

When IRQSIGEN[CINTIEN] is cleared, the eSDHC clears the interrupt request to the host system. The host driver should clear this bit before servicing the SDIO interrupt and should set this bit again after all interrupt requests from the card are cleared to prevent inadvertent interrupts.

If enabled by IRQSTATEN[CINTSEN], the IRQSTAT[CINT] bit can only be cleared by resetting the SDIO interrupt source and then writing one to this bit. Merely writing to this bit has no effect.

In 1-bit mode, the eSDHC detects the SDIO interrupt with or without SD clock (to support wakeup). In 4-bit mode, the interrupt signal is sampled during the interrupt period, so there are some sample delays between the interrupt signal from the SDIO card and the interrupt to the host system interrupt controller. When IRQSTAT[CINT] is set and the host driver needs to start this interrupt service, IRQSTATEN[CINTSEN] is cleared in order to clear IRQSTAT[CINT] that is latched in the eSDHC and to stop driving the interrupt signal to the processor's interrupt controller. The host driver must issue a CMD52 to clear the interrupts at the card. After completion of the card interrupt service, IRQSTATEN[CINTSEN] is set, and the eSDHC can start sampling the interrupt signal again.

See the following illustrations:

- [Figure 11-25](#) (a) for an illustration of the SDIO card interrupt scheme

- Figure 11-25 (b) for the sequences of software and hardware events that take place during card interrupt handling procedure

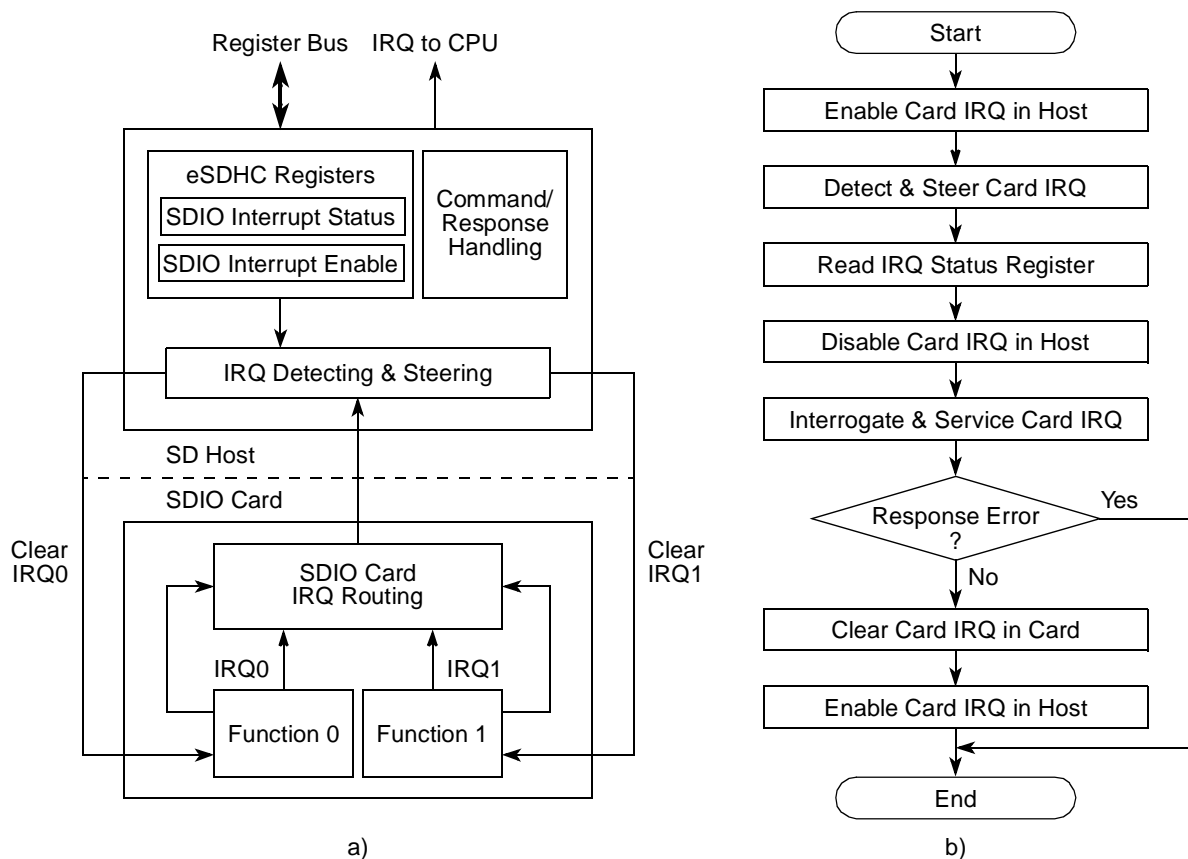


Figure 11-25. a) Card Interrupt Scheme; b) Card Interrupt Detection and Handling Procedure

### 11.5.7 Card Insertion and Removal Detection

The eSDHC uses the  $\overline{SD\_DAT[3]}$  pin or the  $\overline{SD\_CD}$  pin to detect card insertion or removal. When  $\overline{SD\_DAT[3]}$  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  $\overline{SD\_DAT[3]}$  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  $\overline{SD\_DAT[3]}$  pin and generates an interrupt.

When  $\overline{SD\_DAT[3]}$  pin is not used for card detection,  $\overline{SD\_CD}$  must be connected for card detection. It may be implemented by a GPIO. Whether  $\overline{SD\_DAT[3]}$  is configured for card detection or not,  $\overline{SD\_CD}$  is always a reference for card detection, either  $\overline{SD\_DAT[3]}$  or  $\overline{SD\_CD}$  reports card inserted, the eSDHC informs the host system that a card is inserted, and the interrupt is sent if it is enabled.

### 11.5.8 Power Management

When there is no operation between eSDHC and the card through SD bus, the internal clocks in the chip level clock control module can be completely disabled to save power. When eSDHC is needed to

communicate with the card, it can enable the clock and start the operation. This can be done by clearing the SCCR[SDHCCM] bits.

## 11.6 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. Note that eSDHC supports only one SDIO card.

Broadcast commands are intended for all cards, such as GO\_IDLE\_STATE, SEND\_OP\_COND and ALL\_SEND\_CID. In broadcast mode, all cards are in the open-drain mode to avoid bus contention. For the commands of bc and bcr categories, see [Section 11.6.5, “Commands for MMC/SD/SDIO.”](#)

After the broadcast command CMD3 is issued, the cards enter standby mode. Addressed type commands are used from this point. In this mode, the SD\_CMD/SD\_DAT I/O pads turn to push-pull mode, to have the driving capability for maximum frequency operation. For the commands of ac and adtc categories, see [Section 11.6.5, “Commands for MMC/SD/SDIO.”](#)

### 11.6.1 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.

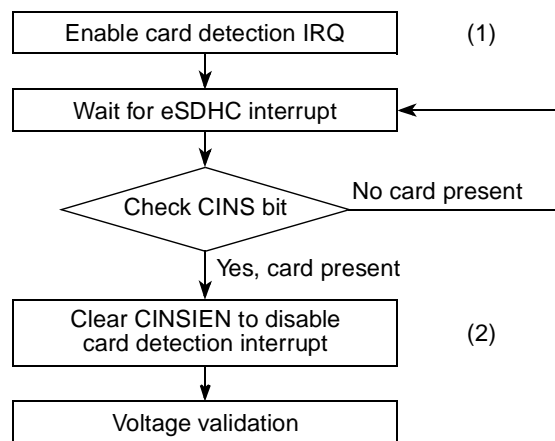
For some scenarios, the response timeout is expected. For instance, after all cards respond to CMD3 and go to the standby state, no response to the host when CMD2 is sent. The host driver should manage false errors similar to this with caution.

## 11.6.2 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.

### 11.6.2.1 Card Detect

See [Figure 11-26](#) for a flow diagram showing the detection of MMC, SDIO, and SD cards using the eSDHC.



**Figure 11-26. 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.

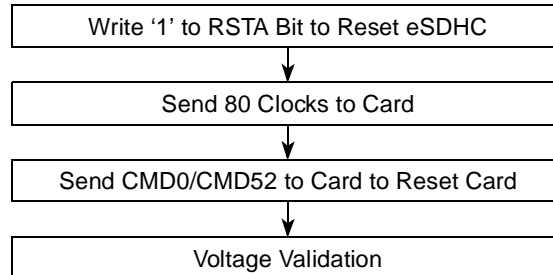
### 11.6.2.2 Reset

The host consists of the following 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. For an SDIO card, CMD52 is used to write I/O reset in CCCR.

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 11-27](#) for the software flow to reset the eSDHC and card.



**Figure 11-27. 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 SD_CLK of frequency around 400 KHz
    poll PRSTAT[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>);
}
  
```

### 11.6.2.3 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 this 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)
- IO\_SEND\_OP\_CONT (CMD5 for SDIO).

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.

The following steps illustrate how to perform voltage validation when a card is inserted:

```

voltage_validation(voltage_range_argument)
{
    label the card as UNKNOWN;
    send_command(IO_SEND_OP_COND, 0x0, <other parameters are omitted>);
        // CMD5, check SDIO operation voltage, command argument is zero
    if (RESP_TIMEOUT != wait_for_response(IO_SEND_OP_COND)) { // SDIO command is accepted
        if (0 < number of IO functions) {
            label the card as SDIO;
            IORDY = 0;
            while (!(IORDY in IO OCR response)) { // set voltage range for each IO function
                send_command(IO_SEND_OP_COND, <voltage range>, <other parameter>);
                wait_for_response(IO_SEND_OP_COND);
            } // end of while ...
        } // end of if (0 < ...)
        if (memory part is present inside SDIO card) label the card as SDCombo;
            // this is an SD-Combo card
    } // end of if (RESP_TIMEOUT...)
    if (the card is labeled as SDIO card) return;
        // card type is identified and voltage range is set, so exit the function;
    send_command(APP_CMD, 0x0, <other parameters are omitted>);
        // CMD55, application specific CMD prefix
    if (no error calling wait_for_response(APP_CMD, <...>)) { // CMD55 is accepted
        send_command(SD_APP_OP_COND, <voltage range>, <...>);
            // ACMD41, to set voltage range for memory part or SD card
        wait_for_response(SD_APP_OP_COND); // voltage range is set
        if (card type is UNKNOWN) label the card as SD;
        return;
    } // end of if (no error ...)
    else if (errors other than timeout occur) { // command/response pair is corrupted
        respond to it by program specific manner;
    } // of else if (response timeout)
    else { // CMD55 is refused, it must be MMC or CE-ATA card
        if (card is already labeled as SD Combo) { // change label
            re-label the card as SDIO;
            ignore the error or report it;
            return; // card is identified as SDIO card
        } // of if (card is ...)
        send_command(SEND_OP_COND, <voltage range>, <...>);
        if (RESP_TIMEOUT == wait_for_response(SEND_OP_COND)) {
            // CMD1 is not accepted, either
            label the card as UNKNOWN;
            return;
        } // of if (RESP_TIMEOUT...)
        if (check for CE-ATA signature succeeded) { // the card is CE-ATA
            store CE-ATA specific info from the signature;
            label the card as CE-ATA;
        } // of if (check for CE-ATA...)
        else label the card as MMC;
    } // of else
}

```

### 11.6.2.4 Card Registry

Card registry on MMC and SD/SDIO/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 SD\_CMD line output drivers 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 SD\_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.

```
card_registry()
{
do { // decide RCA for each card until response timeout
    if(card is labeled as SD Combo or SDIO) { // for SDIO card like device
        send_command(SET_RELATIVE_ADDR, 0x00, <...>);
        // ask SDIO card to publish its RCA
        retrieve RCA from response;
    } // end if (card is labeled as SD Combo...)
    else if (card is labeled as SD) { // for SD card
        send_command(ALL_SEND_CID, <...>);
        if (RESP_TIMEOUT == wait_for_response(ALL_SEND_CID)) break;
    }
} while (1);
}
```

```

        send_command(SET_RELATIVE_ADDR, <...>);
        retrieve RCA from response;
    } // else if (card is labeled as SD ...)
    else if (card is labeled as MMC or CE-ATA) { // treat CE-ATA as MMC
        send_command(ALL_SEND_CID, <...>);
        rca = 0x1; // arbitrarily set RCA, 1 here for example, this RCA is also the
                // relative address to access the CE-ATA card
        send_command(SET_RELATIVE_ADDR, 0x1 << 16, <...>);
                // send RCA at upper 16 bits
    } // end of else if (card is labeled as MMC...)
} while (response is not timeout);
}

```

### 11.6.3 Card Access

These sections describe the supported access modes with external cards.

#### 11.6.3.1 Block Write

This section describes the process of writing data to external cards in block mode.

##### 11.6.3.1.1 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 SD\_DAT line. 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 SD\_DAT line low. The host may poll the status of the card with a SEND\_STATUS command (CMD13) or other means for SDIO 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 SD\_DAT line without interrupting the write operation. When re-selecting the card, it reactivates the busy indication by pulling SD\_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)
  - SDIO cards or the I/O portion of SD Combo cards — IO\_RW\_DIRECT (CMD52) to set I/O block size bit field in the CCCR register (for function 0) or FBR (for functions 1–7)
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.

#### 11.6.3.1.2 Write with Pause

The write operation can be paused during the transfer. Instead of stopping the SD\_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 SD\_DAT0 is not required to de-assert to release busy state, no suspend command is needed.

Similar to the flow described in [Section 11.6.3.1.1, “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)
  - SDIO cards or the I/O portion of SD Combo cards — use IO\_RW\_DIRECT (CMD52) to set the I/O block size bit field in CCCR register (for function 0) or FBR (for functions 1–7)
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 write 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. Therefore, avoid using the suspend command for the SDIO card. When such command is sent, the eSDHC assumes the system switches to another function of the SDIO card and flushes the data buffer. 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.

### 11.6.3.2 Block Read

This section discusses normal read and read with pause.

#### 11.6.3.2.1 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 blocks 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)
  - SDIO cards or the I/O portion of SD Combo cards — use IO\_RW\_DIRECT (CMD52) to set IO block size bit field in CCCR register (for function 0) or FBR (for functions 1–7)
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.

### 11.6.3.2.2 Read with Pause

In general, the read operation is not able to pause. Only the SDIO card (and SD Combo card working under I/O mode) supporting the read wait feature can pause during the read operation. If the SDIO card supports read wait (CCCR[SRW] = 1), the driver can set PROCTL[SABGREQ] to pause the transfer between the data blocks. Before setting SABGREQ, PROCTL[RWCTL] must be set. Otherwise, the eSDHC does not assert the read wait signal during the block gap and data corruption occurs. It is recommended to set the RWCTL bit once the read wait capability of the SDIO card is recognized.

Similar to the flow described in [Section 11.6.3.2.1, “Normal Read,”](#) the read with pause is shown with the same type of read operations:

1. Check CCCR[SRW] in the SDIO card to confirm the card supports read wait.
2. Set PROCTL[RWCTL].
3. Check the card status and wait until the card is ready for data.
4. Set the card block length.
  - MMC/SD cards — use SET\_BLOCKLEN (CMD16)
  - SDIO cards or the I/O portion of SD Combo cards — use IO\_RW\_DIRECT (CMD52) to set IO block size bit field in CCCR register (for function 0) or FBR (for functions 1–7)
5. Set eSDHC BLKATTR[BLKSIZE] to the same as the block length set in the card in Step 2.
6. Set eSDHC BLKATTR[BLKCNT] with the number of blocks to send.
7. 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].
8. Set PROCTL[SABGREQ].
9. Wait for the transfer complete interrupt.
10. Clear PROCTL[SABGREQ].
11. Check the status bit to see if a read CRC error occurred.
12. Set PROCTL[CREQ] to continue the read operation.
13. Wait for the transfer complete interrupt.
14. 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.

### 11.6.3.3 Transfer Error

This section discusses the following errors:

- CRC
- Internal DMA
- Auto CMD12

#### 11.6.3.3.1 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.

#### 11.6.3.3.2 Internal DMA Error

During the data transfer with the internal DMA, if the DMA engine encounters an error on the CSB 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. Depending on initial value of this field and block size, the start address of the corrupted block can be obtained.
- 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.

#### 11.6.3.3.3 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 with one of the following actions (as appropriate to kind of error):

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.

#### 11.6.3.4 Card Interrupt

The external cards can inform the host controller through the use of special signals. For SDIO cards, it can be the low level on the SD\_DAT[1] line during a specific period. It is possible some other external interrupt behaviors can be defined. The eSDHC only monitors the SD\_DAT[1] line and supports SDIO interrupts.

When an SDIO interrupt is captured by the eSDHC and the host system is informed by the eSDHC asserting its interrupt line, the interrupt service of host driver is requested.

As the interrupt source is controlled by the external card, the interrupt from the SDIO card must be served before the CINT bit is cleared. For the card interrupt handling flow, see [Section 11.5.6.3, “Card Interrupt Handling.”](#)

#### 11.6.4 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 SDIO cards, the high speed mode is enabled by writing to CCCR[EHS] after the CCCR[SHS] bit is confirmed. 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 (for SDIO card, by setting RES bit in CCCR register; for other cards, by issuing CMD0), 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.



### 11.6.4.1 Query, Enable and Disable SDIO High Speed Mode

The following pseudo code shows enabling and disabling the high speed mode for SDIO using CMD52.

```
enable_sdio_high_speed_mode(void)
{
    send CMD52 to query bit SHS at address 0x13;
    if (SHS bit is '0')
    {
        report the SDIO card does not support high speed mode and return;
    }
    send CMD52 to set bit EHS at address 0x13 and read after write to confirm EHS bit is set;
    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_sdio_high_speed_mode(void)
{
    send CMD52 to clear bit EHS at address 0x13 and read after write to confirm EHS bit is cleared;
    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)
}
```

### 11.6.4.2 Query, Enable and Disable SD High Speed Mode

The following pseudo code shows enabling and disabling the high speed mode for SD card using CMD6.

```
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)
}
```

### 11.6.4.3 Query, Enable and Disable MMC High Speed Mode

The following pseudo code shows enabling and disabling the high speed mode for MMC using CMD6.

```
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)
}
```

### 11.6.4.4 Set MMC Bus Width

The following pseudo code shows how to set the bus width for MMC using CMD6.

```
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)
}
```

## 11.6.5 Commands for MMC/SD/SDIO

See [Table 11-26](#) for the list of commands for the MMC/SD/SDIO cards. Refer to the corresponding specifications for details about the command information.

Four kinds of commands control the MMC, as follows:

- Broadcast commands (bc)—no response
- Broadcast commands with response (bcr)—response from all cards simultaneously
- Addressed (point-to-point) commands (ac)—no data transfer on SD\_DAT
- Addressed (point-to-point) data transfer commands (ADTC)

**Table 11-26. Commands for MMC/SD/SDIO**

CMD INDEX	Type	Argument	Resp	Abbreviation	Description <sup>1</sup>
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 SD_CMD line.
CMD2	bcr	[31–0] stuff bits	R2	ALL_SEND_CID	Asks all cards to send their CID numbers on the SD_CMD line.
CMD3 <sup>(1)</sup>	ac	[31–0] RCA	R1 R6(SDIO)	SET/SEND_RELATIVE_ADDR	Assigns relative address to the card.
CMD4	bc	[31–16] DSR [15–0] stuff bits	—	SET_DSR	Programs the DSR of all cards.
CMD5	bc	[31–0] OCR without busy	R4	IO_SEND_OP_COND	Asks all SDIO cards in idle state to send their operation conditions register contents in the response on the SD_CMD line.
CMD6 <sup>(2)</sup>	adtc	[31] Mode 0– Check function 1– Switch function [30–8] Reserved for function groups 6 ~ 3 (All 0 or 0xFFFF) [7–4] Function group1 for command system [3–0] Function group2 for access mode	R1	SWITCH_FUNC	Checks switch ability (mode 0) and switch card function (mode 1). Refer to <i>SD Physical Specification version 1.1</i> for details.
CMD6 <sup>(3)</sup>	ac	[31–26] Set to 0 [25–24] Access [23–16] Index [15–8] Value [7–3] Set to 0 [2–0] Cmd Set	R1b	SWITCH	Switches the mode of operation of the selected card or modifies the EXT_CSD registers. Refer to the <i>MultiMediaCard System Specification version 4.0 final draft 2</i> for details.

Table 11-26. Commands for MMC/SD/SDIO (continued)

CMD INDEX	Type	Argument	Resp	Abbreviation	Description <sup>1</sup>
CMD7	ac	[31–16] RCA [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	[31–16] RCA [15–0] stuff bits	R2	SEND_CSD	Addressed card sends its card-specific data (CSD) on the SD_CMD line.
CMD10	ac	[31–16] RCA [15–0] stuff bits	R2	SEND_CID	Addressed card sends its card-identification (CID) on the SD_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	[31–16] RCA [15–0] stuff bits	R1	SEND_STATUS	Addressed card sends its status register.
CMD14	Reserved				
CMD15	ac	[31–16] RCA [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				
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_TRANSMISSION command is received.
CMD21–23	Reserved				

Table 11-26. Commands for MMC/SD/SDIO (continued)

CMD INDEX	Type	Argument	Resp	Abbreviation	Description <sup>1</sup>
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_START	Sets the address of the first erase group within a range to be selected for erase.
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.

Table 11-26. Commands for MMC/SD/SDIO (continued)

CMD INDEX	Type	Argument	Resp	Abbreviation	Description <sup>1</sup>
CMD38	ac	[31–0] stuff bits	R1b	ERASE	Erase all previously selected sectors.
CMD39	ac	[31–16] RCA [15] register write flag [14–8] register address [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 [15–0] stuff bits	R1	APP_CMD	Indicates to the card that the next command is an application specific command rather than a standard command.
CMD56	adtc	[31–1] stuff bits [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.
ACMDs should be preceded with the APP_CMD command (Commands listed below are for SD cards only. Other SD commands not listed below are not supported by this module)					
ACMD6	ac	[31–2] stuff bits [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.

Table 11-26. Commands for MMC/SD/SDIO (continued)

CMD INDEX	Type	Argument	Resp	Abbreviation	Description <sup>1</sup>
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 [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 SD_CMD line.
ACMD42	ac	—	R1	SET_CLR_CARD_DETECT	—
ACMD51	adtc	[31–0] stuff bits	R1	SEND_SCR	Reads the SD Configuration Register (SCR)

<sup>1</sup> Registers mentioned in this table are SD card registers.

#### NOTE

- CMD3 differs for MMC and SD cards  
For MMCs, 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 MMCs and high-speed SD cards. Command SWITCH\_FUNC is used for high speed SD cards.
- Command SWITCH is for high-speed MMCs. 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 11-27](#):

Table 11-27. 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.

## 11.7 Software Restrictions

This section discusses the software restrictions.

### 11.7.1 Initialization Active

The driver should not set INITA bit in System Control register when any of the command line or data lines is active, so the driver should ensure both CDIHB and CIHB bits are cleared. In order to auto clear the INITA bit, the SDCLKEN bit must be '1', otherwise no clocks can go out to the card and INITA never clears.

### 11.7.2 Software Polling Procedure

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.

### 11.7.3 Suspend Operation

In order to suspend the data transfer, the software must inform eSDHC that the suspend command is successfully accepted. To achieve this, after the Suspend command is accepted by the SDIO card, software must send another normal command marked as suspend command (CMDTYP bits set as '01') to inform eSDHC that the transfer is suspended.

If software needs resume the suspended transfer, it should read the value in BLKCNT register to save the remained number of blocks before sending the normal command marked as suspend, otherwise on sending such 'suspend' command, eSDHC regards the current transfer as aborted and change BLKCNT register to its original value, instead of keeping the remained number of blocks.

### 11.7.4 Data Port Access

When the internal DMA is not enabled and a write transaction is in operation, DATPORT (described in [Section 11.4.6, "Buffer Data Port Register \(DATPORT\)"](#)) must not be read. DATPORT also must not be used to read (or write) data by the CPU if the data will be written (or read) by the eSDHC internal DMA.

### 11.7.5 Multi-block Read

For pre-defined multi-block read operation, that is, the number of blocks to read has been defined by previous CMD23 for MMC, or pre-defined number of blocks in CMD53 for SDIO/SDCombo, or whatever multi-block read without abort command at card side, soft reset for data is required by eSDHC to drive the internal state machine to idle mode. Then use reset mechanism. For information on reset mechanism, see Section 3-10, "Error Recovery" in *SD Host Controller Specifications, Ver 2.0 (Jan 2007)*.



# Chapter 12

## DMA Controller (DMAC)

The direct memory access (DMA) is a second-generation platform module capable of performing complex data transfers with minimal intervention from a host processor through 16 programmable channels. The hardware micro-architecture includes a DMA engine, which performs source and destination address calculations, and the actual data movement operations, along with a local memory containing the transfer control descriptors (TCD) for the channels.

Figure 12-1 shows the DMA block diagram.

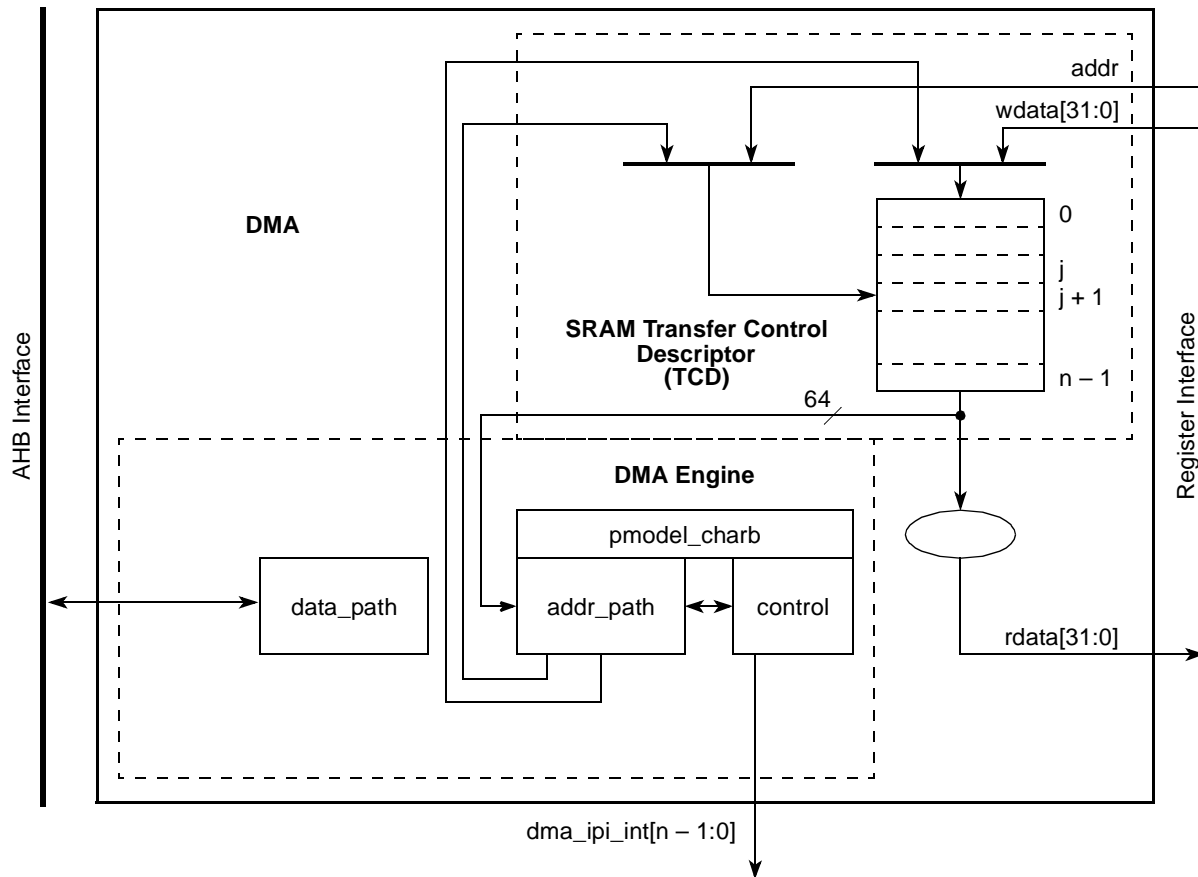


Figure 12-1. DMA Block Diagram

### 12.1 Overview

The DMA is a highly-programmable data transfer engine, which has been optimized to minimize the required intervention from the host processor. It is intended for use in applications where the data size to

be transferred is statically known, and is not defined within the data packet itself. The DMA hardware supports:

- 16 Channels
- 32-byte transfer control descriptor per channel stored in local memory
- 32 bytes of data registers, used as temporary storage to support burst transfers

Throughout this section,  $n$  is used to reference the channel number. Additionally, data sizes are defined as byte (8-bit), half-word (16-bit), word (32-bit), and double word (64-bit).

### 12.1.1 Features

The DMA module supports the following features:

- All data movement via dual-address transfers: read from source and write to destination
  - Programmable source, destination addresses, transfer size, plus support for enhanced addressing modes
- Transfer control descriptor organized to support two-deep, nested transfer operations
  - An *inner* data transfer loop defined by a “minor” byte transfer count
  - An *outer* data transfer loop defined by a “major” iteration count
- Channel service request via one of two methods:
  - Explicit software initiation
  - Initiation via a channel-to-channel linking mechanism for continuous transfers
    - Independent channel linking at end of minor loop and/or major loop

For both the methods, one service request per execution of the minor loop is required

- Support for fixed-priority and round-robin channel arbitration
- Channel completion reported via optional interrupt requests
  - One interrupt per channel, optionally asserted at completion of major iteration count
  - Error terminations are optionally enabled per channel, and logically summed together to form a small number of error interrupt outputs
- Support for scatter/gather DMA processing

## 12.2 DMAC Memory Map/Register Definition

The DMA programming model is partitioned into two sections, both mapped into the IPIslave space: the first region defines a number of registers providing control functions while the second region corresponds to the local transfer control descriptor memory. Reading an unimplemented register bit or memory location returns the value of zero. Read modified write should be used for unimplemented register bits. Any access

to a reserved memory location results in a bus error. Reserved memory locations are indicated in the memory map. [Table 12-1](#) is a 32-bit view of the DMA memory map.

**Table 12-1. DMAC Register Summary**

Offset	Register	Access	Reset	Section/Page
<b>Block Base Address: 0x2_C000</b>				
0x000	DMACR—DMA Control Register	R/W	0x0000_E400	<a href="#">12.2.1/12-3</a>
0x004	DMAES—DMA Error Status Register	R	0x0000_0000	<a href="#">12.3/12-6</a>
0x008– 0x010	Reserved	—	—	—
0x014	DMAEEI—DMA enable error interrupt register	R/W	0x0000_0000	<a href="#">12.3.1/12-8</a>
0x018– 0x019	Reserved	—	—	—
0x01A	DMASEEI—DMA Set Enable Error Interrupt	R/W	0x0000	<a href="#">12.3.2/12-9</a>
0x01B	DMACEEI—DMA Clear Enable Error Interrupt	R/W	0x0000	<a href="#">12.3.3/12-9</a>
0x01C	DMACINT—DMA Clear Interrupt Request	R/W	0x0000	<a href="#">12.3.4/12-10</a>
0x01D	DMACERR—DMA Clear Error	R/W	0x0000	<a href="#">12.3.5/12-11</a>
0x01E	DMASSRT—DMA Set START Bit	R/W	0x0000	<a href="#">12.3.6/12-11</a>
0x01F	DMACDNE—DMA Clear DONE Status Bit	R/W	0x0000	<a href="#">12.3.7/12-12</a>
0x020	Reserved	—	—	—
0x024	DMAINT—DMA interrupt request register	w1c	0x0000_0000	<a href="#">12.3.8/12-12</a>
0x028	Reserved	—	—	—
0x02C	DMAERR—DMA error register	w1c	0x0000_0000	<a href="#">12.3.9/12-13</a>
0x030– 0x034	Reserved	—	—	—
0x038	DMAGPOR—DMA general purpose output register	R/W	0x0000_0000	<a href="#">12.3.10/12-14</a>
0x03C– 0x0FC	Reserved	—	—	—
0x100– 0x13C	DCHPRI <sub>n</sub> —DMA Channel <i>n</i> Priority Register	R/W	0x0000_ <i>nnnn</i>	<a href="#">12.3.11/12-15</a>
0x140– 0xFFC	Reserved	—	—	—

## 12.2.1 DMA Control Register (DMACR)

The 32-bit DMACR defines the basic operating configuration of the DMA. There is a single group of DMA channels labeled as, “group 0,” which contain channel numbers 0–15.

Arbitration within this group can be configured to use either a fixed priority or a round robin. In fixed priority arbitration, the highest priority channel requesting service is selected to execute. The priorities are assigned by the channel priority registers (see [Section 12.3.11, “DMA Channel \*n\* Priority \(DCHPRI<sub>n</sub>\), \*n\*](#)

= 0–15”). In round robin arbitration mode, the channel priorities are ignored and the channels within this group are cycled through without regard to priority.

Minor loop offsets are address offset values added to the final source address (saddr) or destination address (daddr) upon minor loop completion. When minor loop offsets are enabled, the minor loop offset (mloff) is added to the final source address (saddr), or the final destination address (daddr), or both prior to the addresses being written back into the TCD. If the major loop is complete, the minor loop offset is ignored and the major loop address offsets (slast and dlast\_sga) are used to compute the next saddr and daddr values.

When minor loop mapping is enabled (DMACR[EMLM] = 1), TCD word2 is redefined. A portion of TCD word2 is used to specify multiple fields: an source enable bit (smloe) to specify the minor loop offset should be applied to the source address (saddr) upon minor loop completion, an destination enable bit (dmloe) to specify the minor loop offset should be applied to the destination address (daddr) upon minor loop completion, and the sign extended minor loop offset value (mloff). The same offset value (mloff) is used for both source and destination minor loop offsets. When either minor loop offset is enabled (smloe set or dmloe set), the nbytes field is reduced to 8 bits. When both minor loop offsets are disabled (smloe cleared and dmloe cleared), the nbytes field becomes a 30-bit vector.

Figure 12-2 shows the DMA control register.

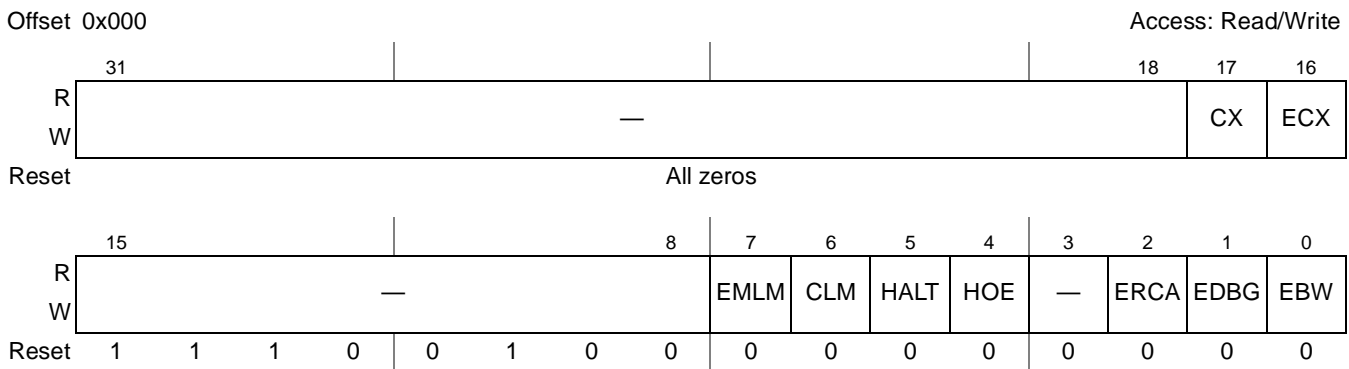


Figure 12-2. DMA Control Register (DMACR)

Table 12-2 describes the DMACR fields.

Table 12-2. DMA Control Register (DMACR) Field Descriptions

Bits	Name	Description
31–18	—	Reserved
17	CX	Cancel transfer. 0 Normal operation. 1 Cancel the remaining data transfer. Stop the executing channel and force the minor loop to be finished. The cancel takes effect after the last write of the current read/write sequence. The CX bit clears itself after the cancel has been honored. This cancel retires the channel normally as if the minor loop was completed.

Table 12-2. DMA Control Register (DMACR) Field Descriptions (continued)

Bits	Name	Description
16	ECX	Error cancel transfer. 0 Normal operation. 1 Cancel the remaining data transfer in the same fashion as the CX cancel transfer. Stop the executing channel and force the minor loop to be finished. The cancel takes effect after the last write of the current read/write sequence. The ECX bit clears itself after the cancel has been honored. In addition to cancelling the transfer, the ECX treats the cancel as an error condition; thus updating the DMAES register and generating an optional error interrupt (see <a href="#">Section 12.3, "DMA Error Status (DMAES)"</a> ).
15–8	—	Reserved
7	EMLM	Enable minor loop mapping. 0 Minor loop mapping disabled. TCDn.word2 is defined as a 32-bit nbytes field. 1 Minor loop mapping enabled. When set, TCDn.word2 is redefined to include individual enable fields, an offset field and the nbytes field. The individual enable fields allow the minor loop offset to be applied to the source address, the destination address, or both. The nbytes field is reduced when either offset is enabled.
6	CLM	Continuous link mode. 0 A minor loop channel link made to itself will go through channel arbitration before being activated again. 1 A minor loop channel link made to itself will not go through channel arbitration before being activated again. Upon minor loop completion, the channel will active again if that channel has a minor loop channel link enabled and the link channel is itself. This effectively applies the minor loop offsets and restarts the next minor loop.
5	HALT	Halt DMA operations. 0 Normal operation. 1 Stall the start of any new channels. Executing channels are allowed to complete. Channel execution resumes when the HALT bit is cleared.
4	HOE	Halt on error. 0 Normal operation. 1 Any error causes the HALT bit to be set. Subsequently, all service requests will be ignored until the HALT bit is cleared.
3	—	Reserved
2	ERCA	Enable round robin channel arbitration. 0 Fixed priority arbitration is used for channel selection. 1 Round robin arbitration is used for channel selection.
1	EDBG	Enable debug. 0 The assertion of the ipg_debug input is ignored. 1 The assertion of the ipg_debug input causes the DMA to stall the start of a new channel. Executing channels are allowed to complete. Channel execution resumes when either the ipg_debug input is negated or the EDBG bit is cleared.
0	EBW	Enable buffered writes. 0 The bufferable write signal (hprot[2]) is not asserted during AHB writes. 1 The bufferable write signal (hprot[2]) is asserted on all AHB writes except for the last write sequence write sequence.

## 12.3 DMA Error Status (DMAES)

The DMAES register provides information concerning the last recorded channel error. Channel errors can be caused by a configuration error (an illegal setting in the transfer control descriptor or an illegal priority register setting in fixed arbitration mode) or an error termination to a bus master read or write cycle.

A configuration error is caused when the starting source or destination address, source or destination offsets, minor loop byte count and the transfer size represent an inconsistent state. The addresses and offsets must be aligned on 0-modulo-transfer\_size boundaries, and the minor loop byte count must be a multiple of the source and destination transfer sizes. All source reads and destination writes must be configured to the natural boundary of the programmed transfer size respectively. In fixed arbitration mode, a configuration error is caused by any two channel priorities. All channel priority levels within the “group 0” must be unique. If a scatter/gather operation is enabled upon channel completion, a configuration error is reported if the scatter/gather address (dlast\_sga) is not aligned on a 32 byte boundary. If minor loop channel linking is enabled upon channel completion, a configuration error is reported when the link is attempted if the TCD.citer.e\_link bit does not equal the TCD.biter.e\_link bit. All configuration error conditions except scatter/gather and minor loop link error are reported as the channel is activated and assert an error interrupt request, if enabled. A scatter/gather configuration error is reported when the scatter/gather operation begins at major loop completion when properly enabled. A minor loop channel link configuration error is reported when the link operation is serviced at minor loop completion.

If a system bus read or write is terminated with an error, the data transfer is stopped and the appropriate bus error flag set. In this case, the state of the channel's transfer control descriptor is updated by the DMA engine with the current source address, destination address and current iteration count at the point of the fault. When a system bus error occurs, the channel is terminated after the read or write transaction which is already pipelined after errant access, has completed. If a bus error occurs on the last read prior to beginning the write sequence, the write executes using the data captured during the bus error. If a bus error occurs on the last write prior to switching to the next read sequence, the read sequence executes before the channel is terminated due to the destination bus error.

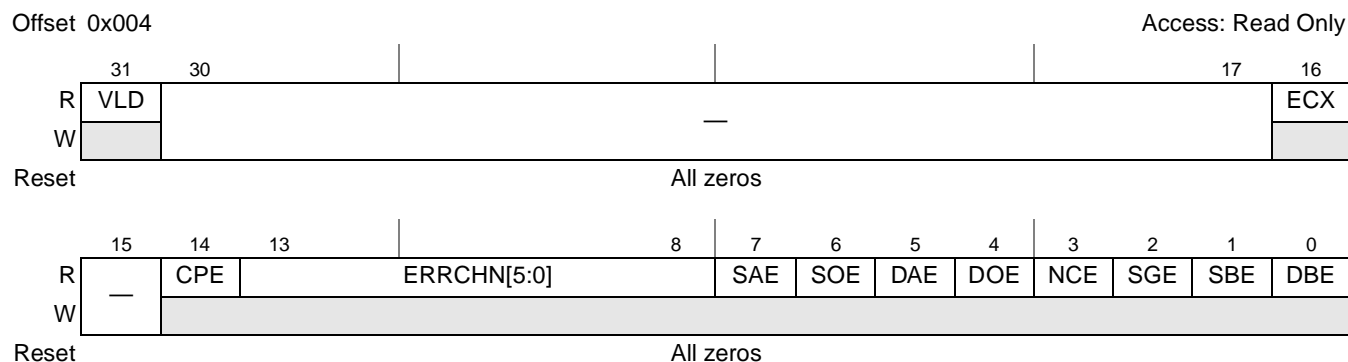
A transfer may be cancelled by software via the DMACR[CX] bit or hardware via the dma\_cancel\_xfer input signal. When a cancel transfer request is recognized, the dma\_engine stops processing the channel. The current read-write sequence is allowed to finish. If the cancel occurs on the last read-write sequence of a major or minor loop, the cancel request is discarded and the channel retires normally.

The error cancel transfer is the same as a cancel transfer except the DMAES register is updated with the cancelled channel number and error cancel bit is set. The TCD of a cancelled channel has the source address and destination address of the last transfer saved in the TCD. It is the responsibility of the user to initialize the TCD again should the channel need to be restarted because the aforementioned fields have been modified by the dma\_engine and no longer represent the original parameters. When a transfer is cancelled via the error cancel transfer mechanism (setting the DMACR[ECX] or asserting the dma\_err\_cancel\_xfer input), the channel number is loaded into the ERRCHN field and the ECX and VLD bits are set in the DMAES register. In addition, an error interrupt may be generated if enabled. See [Section 12.3.9, “DMA Error Register \(DMAERR\),”](#) for error interrupt details.

The occurrence of any type of error causes the DMA engine to immediately stop, and the appropriate channel bit in the DMA error register to be asserted. At the same time, the details of the error condition are loaded into the DMAES register. The major loop complete indicators, setting the transfer control

descriptor done flag and the possible assertion of an interrupt request, are not affected when an error is detected. See [Table 12-3](#) for the DMAES definition.

[Figure 12-3](#) shows the DMA error status register.



**Figure 12-3. DMA Error Status Register (DMAES)**

**Table 12-3. DMAES Field Descriptions**

Bits	Name	Value
31	VLD	Logical OR of all the DMAERR status bits. 0 No DMAERR bits are set. 1 At least one DMAERR bit is set indicating a valid error exists that has not been cleared.
30–17	—	Reserved
16	ECX	Transfer cancelled. 0 No cancelled transfers 1 The last recorded entry was a cancelled transfer via the error cancel transfer input.
15	—	Reserved
14	CPE	Channel priority error. 0 No channel priority error. 1 The last recorded error was a configuration error in the channel priorities. All channel priorities are not unique.
13–8	ERRCHN	Error channel number or cancelled channel number. The channel number of the last recorded error (excluding CPE errors) or last recorded transfer that was error cancelled.
7	SAE	Source address error. 0 No source address configuration error. 1 The last recorded error was a configuration error detected in the <code>TCD.saddr</code> field. <code>TCD.saddr</code> is inconsistent with <code>TCD.ssize</code> .
6	SOE	Source offset error. 0 No source offset configuration error. 1 The last recorded error was a configuration error detected in the <code>TCD.soff</code> field. <code>TCD.soff</code> is inconsistent with <code>TCD.ssize</code> .
5	DAE	Destination address error. 0 No destination address configuration error. 1 The last recorded error was a configuration error detected in the <code>TCD.daddr</code> field. <code>TCD.daddr</code> is inconsistent with <code>TCD.dsize</code> .

Table 12-3. DMAES Field Descriptions (continued)

Bits	Name	Value
4	DOE	Destination offset error. 0 No destination offset configuration error. 1 The last recorded error was a configuration error detected in the <code>TCD.doff</code> field. <code>TCD.doff</code> is inconsistent with <code>TCD.dsize</code> .
3	NCE	Nbytes/citer configuration error. 0 No nbytes/citer configuration error. 1 The last recorded error was a configuration error detected in the <code>TCD.nbytes</code> or <code>TCD.citer</code> fields. <code>TCD.nbytes</code> is not a multiple of <code>TCD.ssize</code> and <code>TCD.dsize</code> , or <code>TCD.citer</code> is equal to zero, or <code>TCD.citer.e_link</code> is not equal to <code>TCD.biter.e_link</code> .
2	SGE	Scatter/gather configuration error. 0 No scatter/gather configuration error. 1 The last recorded error was a configuration error detected in the <code>TCD.dlast_sga</code> field. This field is checked at the beginning of a scatter/gather operation after major loop completion if <code>TCD.e_sg</code> is enabled. <code>TCD.dlast_sga</code> is not on a 32 byte boundary.
1	SBE	Source bus error. 0 No source bus error. 1 The last recorded error was a bus error on a source read.
0	DBE	Destination bus error. 0 No destination bus error. 1 The last recorded error was a bus error on a destination write.

### 12.3.1 DMA Enable Error Interrupt Register (DMAEEI)

DMAEEI provides a bit map for the 16 channels to enable the error interrupt signal for each channel. The state of any given channel's error interrupt enable is directly affected by writes to this register; it is also affected by writes to DMASEEI and DMACEEI. DMASEEI and DMACEEI are provided so that the error interrupt enable for a single channel can easily be modified without the need to perform a read-modify-write sequence to DMAEEI.

Both the DMA error indicator and this error interrupt enable flag must be asserted before an error interrupt request for a given channel is asserted.

Figure 12-4 shows the DMA enable error interrupt register.

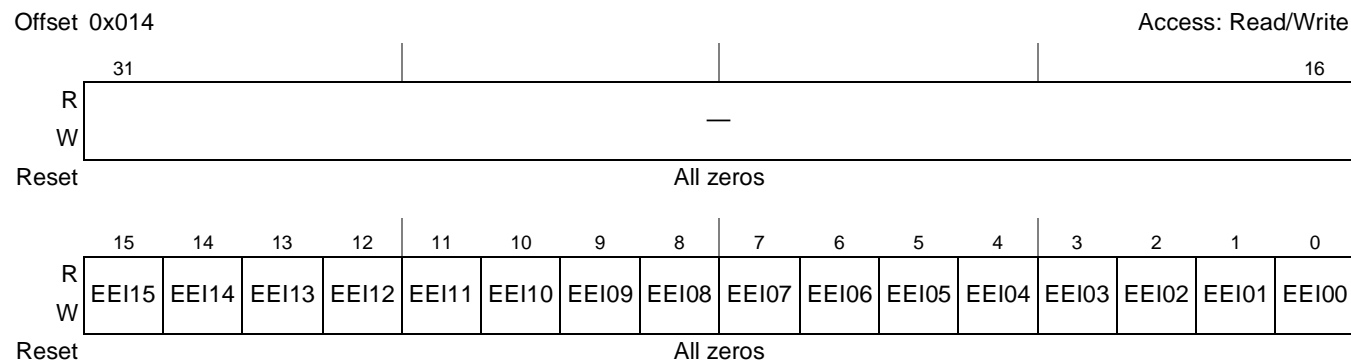


Figure 12-4. DMA Enable Error Interrupt Register (DMAEEI)



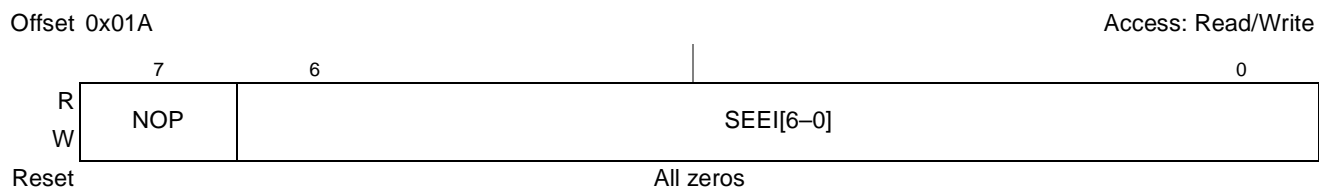
See [Table 12-4](#) for the DMAEEI definition.

**Table 12-4. DMAEEI Field Descriptions**

Bits	Name	Description
31–16	—	Reserved
15–0	EEIn	Enable error interrupt <i>n</i> . 0 The error signal for channel <i>n</i> does not generate an error interrupt. 1 The assertion of the error signal for channel <i>n</i> generate an error interrupt request.

### 12.3.2 DMA Set Enable Error Interrupt (DMASEEI)

DMASEEI, shown in [Figure 12-5](#), provides a simple memory-mapped mechanism to set a given bit in the DMAEEI register to enable the error interrupt for a given channel. The data value on a register write causes the corresponding bit in DMAEEI to be set. A data value of 64 to 127 (regardless of the number of implemented channels) provides a global set function, forcing the entire contents of DMAEEI to be asserted. If bit 7 is set, the command is ignored. This allows multiple-byte registers to be written as a 32-bit word. Reads of this register return all zeros.



**Figure 12-5. DMA Set Enable Error Interrupt Register**

[Table 12-5](#) defines the DMASEEI fields.

**Table 12-5. DMASEEI Field Descriptions**

Bits	Name	Description
7	NOP	No operation. 0 Normal operation. 1 No operation, ignore bits 6–0.
6–0	SEEI	Set enable error interrupt. 0–15 Set the corresponding bit in DMAEEI. 16–63 Reserved 64–127 Set all bits in DMAEEI.

### 12.3.3 DMA Clear Enable Error Interrupt (DMACEEI)

The DMACEEI register, shown in [Figure 12-6](#), provides a simple memory-mapped mechanism to clear a given bit in the DMAEEI register to disable the error interrupt for a given channel. The data value on a register write causes the corresponding bit in DMAEEI to be cleared. A data value of 64 to 127 (regardless of the number of implemented channels) provides a global clear function, forcing the entire contents of DMAEEI to be zeroed, disabling all DMA request inputs. If bit 7 is set, the command is ignored. This allows multiple-byte registers to be written as a 32-bit word. Reads of this register return all zeros.

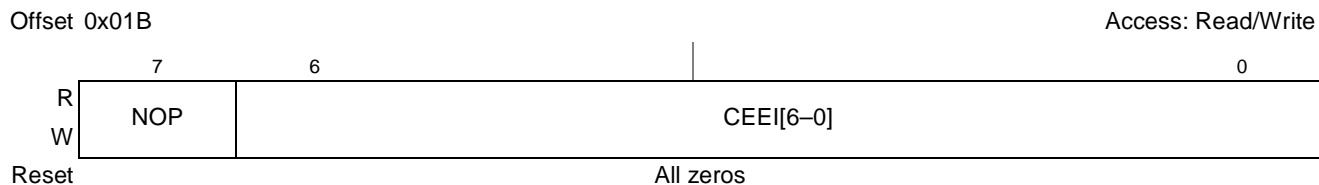


Figure 12-6. DMA Clear Enable Error Interrupt Register

Table 12-6 defines the DMACEEI fields.

Table 12-6. DMACEEI Field Descriptions

Bits	Name	Description
7	NOP	No operation. 0 Normal operation. 1 No operation, ignore bits 6–0.
6–0	CEEI	Clear enable error interrupt. 0–15 Clear corresponding bit in DMAEEI. 16–63 Reserved 64–127 Clear all bits in DMAEEI.

### 12.3.4 DMA Clear Interrupt Request (DMACINT)

DMACINT, shown in Figure 12-7, provides a simple memory-mapped mechanism to clear a given bit in the DMAINT register to disable the interrupt request for a given channel. The given value on a register write causes the corresponding bit in DMAINT to be cleared. A data value of 64 to 127 (regardless of the number of implemented channels) provides a global clear function, forcing the entire contents of DMAINT to be zeroed, disabling all DMA interrupt requests. If bit 7 is set, the command is ignored. This allows multiple-byte registers to be written as a 32-bit word. Reads of this register return all zeros.

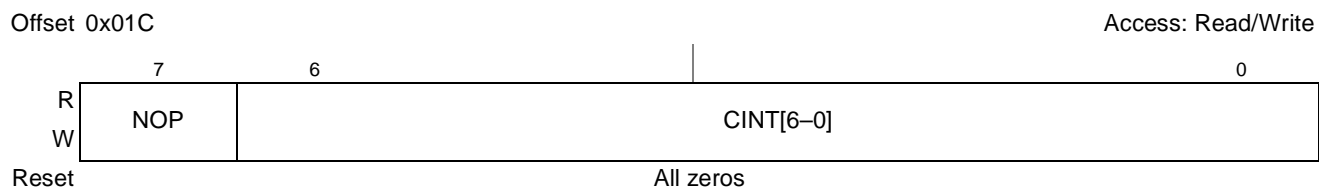


Figure 12-7. DMA Clear Interrupt Request Register

Table 12-7 defines the DMACINT fields.

Table 12-7. DMACINT Field Descriptions

Bits	Name	Description
7	NOP	No operation. 0 Normal operation. 1 No operation, ignore bits 6–0.
6–0	CINT	Clear interrupt request. 0–15 Clear the corresponding bit in DMAINT. 16–63 Reserved 64–127 Clear all bits in DMAINT.

### 12.3.5 DMA Clear Error (DMACERR)

DMACEER, shown in Figure 12-8, provides a simple memory-mapped mechanism to clear a given bit in the DMAERR register to disable the error condition flag for a given channel. The given value on a register write causes the corresponding bit in DMAERR to be cleared. A data value of 64 to 127 (regardless of the number of implemented channels) provides a global clear function, forcing the entire contents of DMAERR to be zeroed, clearing all channel error indicators. If bit 7 is set, the command is ignored. This allows multiple-byte registers to be written as a 32-bit word. Reads of this register return all zeros.



Figure 12-8. DMA Clear Error Register

Table 12-8 defines the DMACERR fields.

Table 12-8. DMACERR Field Descriptions

Bits	Name	Description
7	NOP	No operation. 0 Normal operation. 1 No operation, ignore bits 6–0.
6–0	CERR	Clear error indicator. 0–15 Clear corresponding bit in DMAERR. 16–63 Reserved 64–127 Clear all bits in DMAERR.

### 12.3.6 DMA Set START Bit (DMASSRT)

DMASSRT, shown in Figure 12-9, provides a simple memory-mapped mechanism to set the START bit in the TCD of the given channel. The data value on a register write causes the START bit in the corresponding Transfer Control Descriptor to be set. A data value of 64 to 127 (regardless of the number of implemented channels) provides a global set function, forcing all START bits to be set. If bit 7 is set, the command is ignored. This allows multiple byte registers to be written as a 32-bit word. Reads of this register return all zeros.

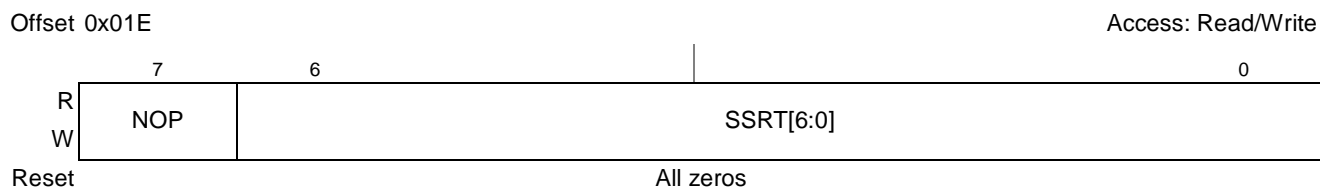


Figure 12-9. DMA Set START Bit Register

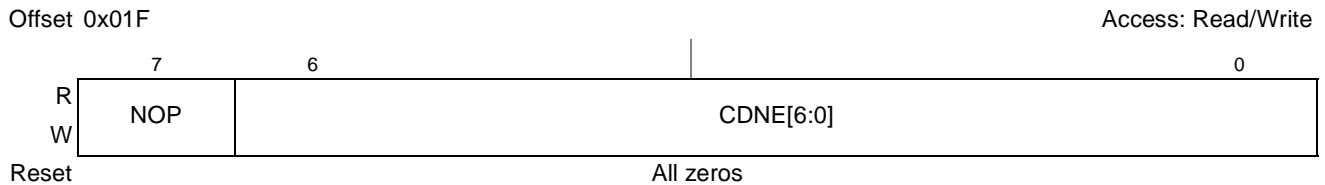
Table 12-9 defines DMASSRT fields.

**Table 12-9. DMASSRT Field Descriptions**

Bits	Name	Description
7	NOP	No operation. 0 Normal operation. 1 No operation, ignore bits 6–0.
6–0	SSRT	Set START bit (channel service request). 0–15 Set the corresponding channel's TCD.start. 16–63 Reserved 64–127 Set all TCD.start bits.

### 12.3.7 DMA Clear DONE Status (DMACDNE)

DMACDNE, shown in Figure 12-10, provides a simple memory-mapped mechanism to clear the DONE bit in the TCD of the given channel. The data value on a register write causes the DONE bit in the corresponding transfer control descriptor to be cleared. A data value of 64 to 127 (regardless of the number of implemented channels) provides a global clear function, forcing all DONE bits to be cleared. If bit 7 is set, the command is ignored. This allows multiple byte registers to be written as a 32-bit word. Reads of this register return all zeros.



**Figure 12-10. DMA Clear DONE Status Register**

Table 12-10 shows the DNACDNE fields.

**Table 12-10. DMACDNE Field Descriptions**

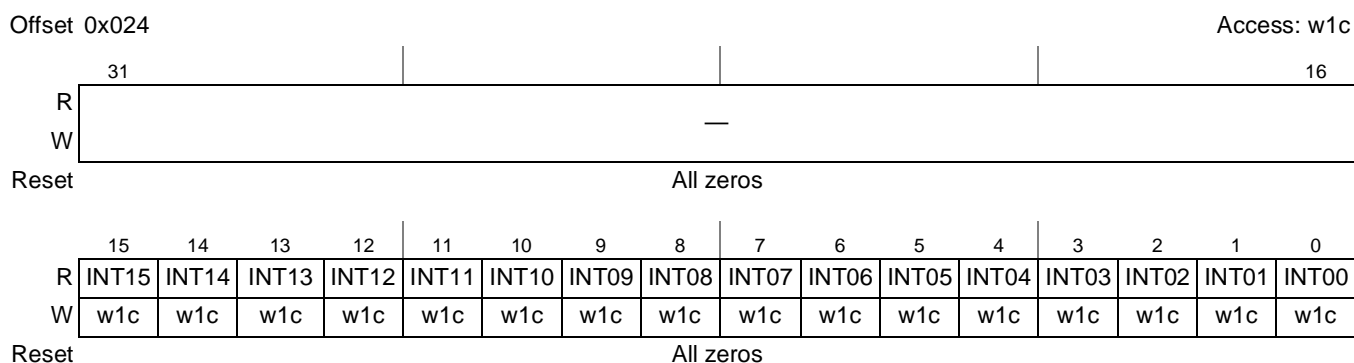
Bits	Name	Description
7	NOP	No operation 0 Normal operation. 1 No operation, ignore bits 6–0.
6–0	CDNE	Clear DONE status bit. 0–15 Clear the corresponding channel's DONE bit. 16–63 Reserved 64–127 Clear all TCD DONE bits.

### 12.3.8 DMA Interrupt Request Register (DMAINT)

DMAINT, shown in Figure 12-11, provide a bit map for the implemented 16 channels, signaling the presence of an interrupt request for each channel. The DMA engine signals the occurrence of a programmed interrupt upon completion of a data transfer as defined in the transfer control descriptor by setting the appropriate bit in this register. The outputs of this register are directly routed to the platform's

interrupt controller. During the execution of the interrupt service routine associated with any given channel, it is software's responsibility to clear the appropriate bit, negating the interrupt request. Typically, a write to the DMACINT register in the interrupt service routine is used for this purpose.

The state of any given channel's interrupt request is directly affected by writes to this register; it is also affected by writes to the DMACINT register. On writes to DMAINT, a one in any bit position clears the corresponding channel's interrupt request. A zero in any bit position has no effect on the corresponding channel's current interrupt status. The DMACINT register is provided so the interrupt request for a single channel can easily be cleared without the need to perform a read-modify-write sequence to DMAINT.



**Figure 12-11. DMA Interrupt Request Register Low (DMAINT)**

Table 12-11 defines the DMAINT fields.

**Table 12-11. DMAINT Field Descriptions**

Bits	Name	Description
31–16	—	Reserved
15–0	INT $n$	DMA interrupt request $n$ (write one to clear) 0 The interrupt request for channel $n$ is cleared. 1 The interrupt request for channel $n$ is active.

### 12.3.9 DMA Error Register (DMAERR)

DMAERR, shown in Figure 12-12, provides a bit map for the 16 channels, signaling the presence of an error for each channel. The DMA engine signals the occurrence of a error condition by setting the appropriate bit in this register. The outputs of this register are enabled by the contents of the DMAEEI register, logically summed across the 16 channels to form the “group 0” error interrupt request, which is then routed to the platform's interrupt controller. During execution of the interrupt service routine associated with any DMA errors, it is software's responsibility to clear the appropriate bit, negating the error interrupt request; typically, a write to the DMACERR register in the interrupt service routine is used for this purpose. Recall that the normal DMA channel completion indicators, setting the transfer control descriptor done flag and the possible assertion of an interrupt request, are not affected when an error is detected.

The contents of this register can also be polled, and a non-zero value indicates the presence of a channel error regardless of the state of the DMAEEI register. The state of any given channel's error indicators is affected by writes to this register; it is also affected by writes to DMACERR. On writes to DMAERR, a

one in any bit position clears the corresponding channel’s error status; a zero in any bit position has no effect. DMACERR is provided so the error indicator for a single channel can easily be cleared.



**Figure 12-12. DMA Error Register (DMAERR)**

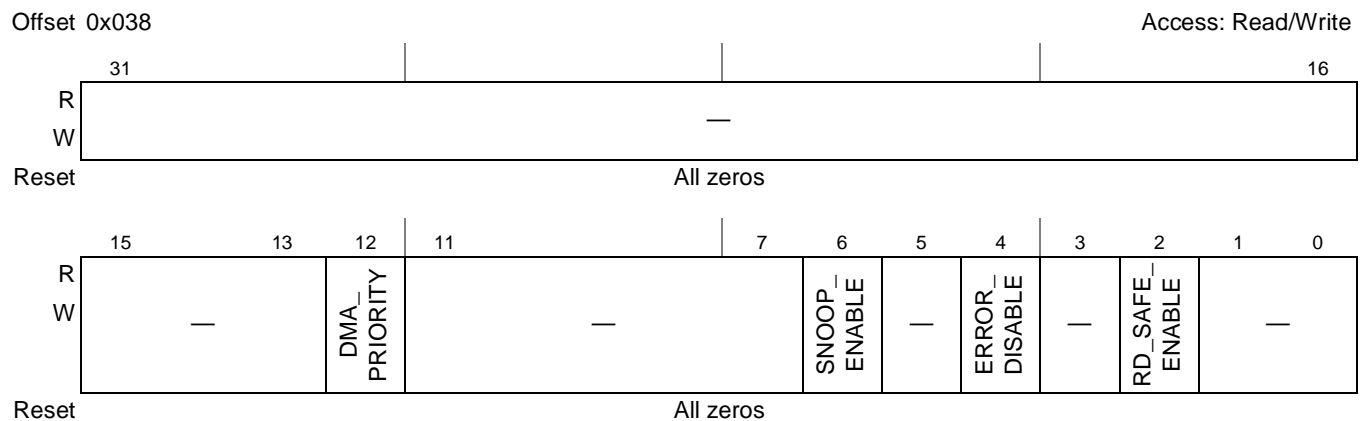
See [Table 12-12](#) for the DMAERR definition.

**Table 12-12. DMAERR Field Descriptions**

Bits	Name	Description
31–16	—	Reserved
15–0	INT $n$	DMA error $n$ (write one to clear) 0 An error in channel $n$ has not occurred. 1 An error in channel $n$ has occurred.

### 12.3.10 DMA General Purpose Output Register (DMAGPOR)

The DMAGPOR register, as shown in [Figure 12-13](#), provides a general purpose register in the programmer’s model that outputs the register contents.



**Figure 12-13. DMA General Purpose Output Register (DMAGPOR)**

See [Table 12-13](#) for the DMAGPOR definition.

**Table 12-13. DMAGPOR Field Descriptions**

Bits	Name	Description
31–13	—	Reserved
12	DMA_PRIORITY	DMA priority. 0 Low priority 1 High priority
11–7	—	Reserved
6	SNOOP_ENABLE	Snoop attribute. 0 DMA transactions are not snooped by e300 CPU data cache 1 DMA transactions are snooped by e300 CPU data cache
5	—	Reserved
4	ERROR_DISABLE	Ignore or react to bus errors. 0 React to bus transaction errors 1 Ignore bus transaction errors
3	—	Reserved
2	RD_SAFE_ENABLE	Read Safe enable. This bit should be set only if the target of read dma operation is a well behaved memory which 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. 0 It is not safe to read more bytes that were intended 1 It is safe to read more bytes that were intended
1–0	—	Reserved

### 12.3.11 DMA Channel *n* Priority (DCHPRI<sub>*n*</sub>), *n* = 0–15

When the fixed-priority channel arbitration mode is enabled (DMACR[ERCA] = 0), the contents of these registers define the unique priorities associated with each channel within the group. The channel priorities are evaluated by numeric value, that is, 0 is the lowest priority, 1 is the next higher priority, then 2, 3, and so on. Software must program the channel priorities with unique values, otherwise a configuration error is reported. The range of the priority value is limited to the values of 0 through 15. When read, the GRPPRI bits of the DCHPRI register reflect the current priority level of the channels. See [Figure 12-2](#) and [Table 12-2](#) for the DMACR definition.

Channel preemption is enabled on a per channel basis by setting the ECP bit in the DCHPRI register. Channel preemption allows the executing channel's data transfers to be temporarily suspended in favor of starting a higher priority channel. Once the preempting channel has completed all of its minor loop data transfers, the preempted channel is restored and resumes execution. After the restored channel completes one read/write sequence, it is again eligible for preemption. If any higher priority channel is requesting service, the restored channel is suspended and the higher priority channel is serviced. Nested preemption (attempting to preempt a preempting channel) is not supported. Once a preempting channel begins execution, it cannot be preempted. Preemption is only available when fixed arbitration is selected for channel arbitration modes.

A channel's ability to preempt another channel can be disabled by setting the DPA bit in the DCHPRI register. When a channel's preempt ability is disabled, that channel cannot suspend a lower priority channel's data transfer; regardless of the lower priority channel's ECP setting. This allows for a pool of low priority, large data moving channels to be defined. These low priority channels can be configured to not preempt each other, thus preventing a low priority channel from consuming the preempt slot normally available a true, high priority channel.

Figure 12-14 shows the DMA clear DONE status register.

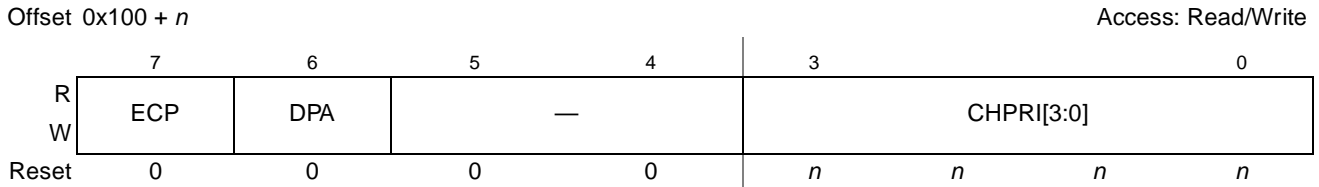


Figure 12-14. DMA Clear DONE Status Register

Table 12-14 defines the DCHPRI fields.

Table 12-14. DCHPRI $n$  Field Descriptions

Bits	Name	Description
7	ECP	Enable channel preemption. 0 Channel $n$ cannot be suspended by a higher priority channel's service request. 1 Channel $n$ can be temporarily suspended by the service request of a higher priority channel.
6	DPA	Disable preempt ability. 0 Channel $n$ can suspend a lower priority channel. 1 Channel $n$ cannot suspend any channel, regardless of channel priority.
5-4	—	Reserved
3-0	CHPRI	Channel $n$ arbitration priority. Channel priority when fixed-priority arbitration is enabled.

### 12.3.12 Transfer Control Descriptor (TCD)

Each channel requires a 32-byte transfer control descriptor for defining the desired data movement operation. The channel descriptors are stored in the local memory in sequential order: channel 0, channel

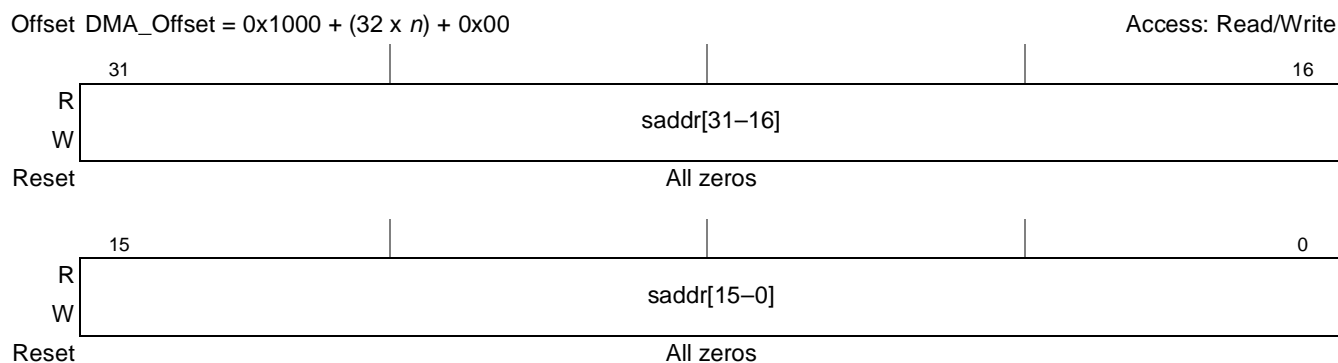


1, ..., channel [n – 1]. The definitions of the TCD are presented as eight 32-bit values. Table 12-15 is a 32-bit view of the basic TCD structure.

**Table 12-15. TCD 32-Bit Memory Structure**

DMA Offset	TCD Field	
$0x1000 + (32 \times n) + 0x000$	Source Address (saddr)	
$0x1000 + (32 \times n) + 0x004$	Transfer Attributes (smod, ssize, dmod, dsize)	Signed Source Address Offset (soff)
$0x1000 + (32 \times n) + 0x008$	Inner Minor Byte Count (nbytes)	
$0x1000 + (32 \times n) + 0x00C$	Last Source Address Adjustment (slast)	
$0x1000 + (32 \times n) + 0x010$	Destination Address (daddr)	
$0x1000 + (32 \times n) + 0x014$	Current Major Iteration Count (citer)	Signed Destination Address Offset (doff)
$0x1000 + (32 \times n) + 0x018$	Last Destination Address Adjustment/Scatter Gather Address (dlast_sga)	
$0x1000 + (32 \times n) + 0x01C$	Beginning Major Iteration Count (biter)	Channel Control/Status (bwc, major.linkch, done, active, major.e_link, e_sg, d_req, int_half, int_maj, start)

Figure 12-15 shows the TCD word 0 field.



**Figure 12-15. TCD Word 0 (TCDn.saddr) Field**

Table 12-16 describes the TCD word 0 fields.

**Table 12-16. TCD Word 0 (TCDn.saddr) Field Description**

Bits	Name	Description
31-0	saddr	Source address. Memory address pointing to the source data.

Figure 12-16 shows the TCD word 1 field.

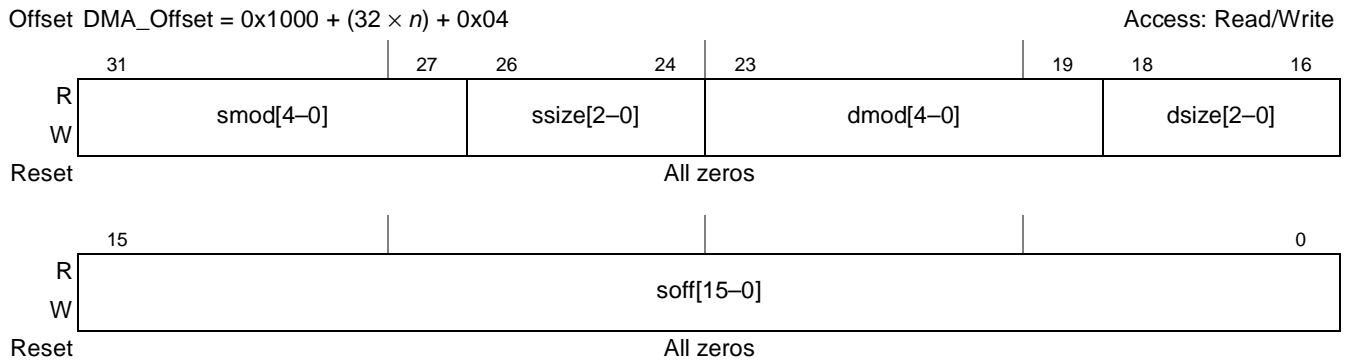


Figure 12-16. TCD Word 1 (TCDn.{soff, smod, ssize, dmod, dsize}) Fields

Table 12-17 describes the TCD word 1 fields.

Table 12-17. TCD Word 1 (TCDn.{smod, ssize, dmod, dsize, soff}) Field Descriptions

Bits	Name	Description
31-27	smod	Source address modulo. 0 Source address modulo feature is disabled. Other The value defines a specific address bit which is selected to be either the value after saddr + soff calculation is performed or the original register value. This feature provides the ability to easily implement a circular data queue. For data queues requiring power-of-2 'size' bytes, the queue should be based at a 0-modulo-size address and the smod field set to the appropriate value to freeze the upper address bits. The bit select is defined as $((1 \ll \text{smod}[4:0]) - 1)$ where a resulting 1 in a bit location selects the next state address for the corresponding address bit location and a 0 selects the original register value for the corresponding address bit location. For this application, the soff is typically set to the transfer size to implement post-increment addressing with the smod function constraining the addresses to a 0-modulo-size range.
26-24	ssize	Source data transfer size. 000 8-bit 001 16-bit 010 32-bit 011 64-bit 100 16-byte 100 Reserved 101 32-byte 110 Reserved 111 Reserved
23-19	dmod	Destination address modulo. See the smod definition.
18-16	dsize	Destination data transfer size. See the ssize definition.
15-0	soff	Source address signed offset. Sign-extended offset applied to the current source address to form the next-state value as each source read is completed.

When minor loop mapping (DMACR[EMLM] = 1) is enabled, TCD word2, shown in Figure 12-17, is redefined as four fields: a source minor loop offset enable, a destination minor loop offset enable, a minor loop offset field, and a nbytes field.

Offset DMA\_Offset = 0x1000 + (32 x n) + 0x08

Access: Read/Write

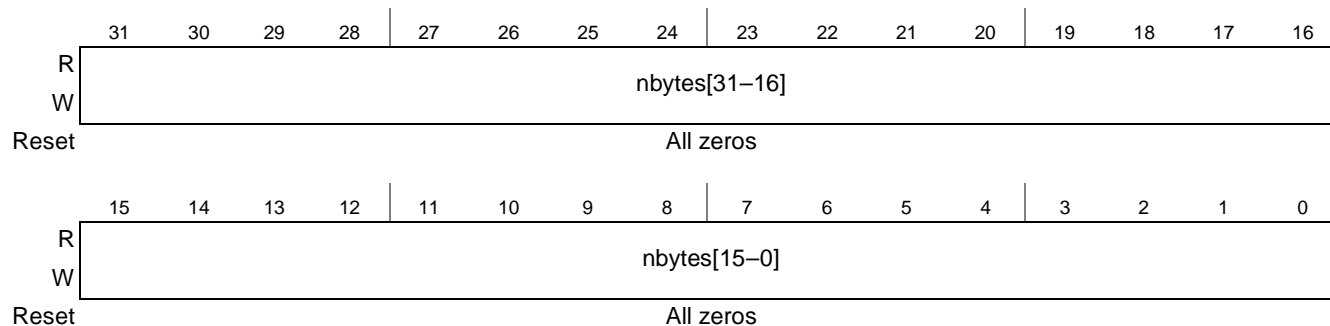


Figure 12-17. TCD Word 2 (TCDn.nbytes) Field

Table 12-18 describes the TCD word 2 fields.

Table 12-18. TCD Word 2 (TCD.nbytes) Field Description

Bits	Name	Description
nbytes[31-0]	nbytes[31-0]	Inner minor byte transfer count. Number of bytes to be transferred in each service request of the channel. As a channel is activated, the contents of the appropriate TCD is loaded into the DMA engine, and the appropriate reads and writes performed until the complete byte transfer count has been transferred. This is an indivisible operation and cannot be stalled or halted. Once the minor count is exhausted, the current values of the saddr and daddr are written back into the local memory, the major iteration count is decremented and restored to the local memory. If the major iteration count is completed, additional processing is performed. The nbytes value 0x0000_0000 is interpreted as 0x1_0000_0000, thus specifying a 4GB transfer.

Figure 12-18 shows the TCD word 3 field.

Offset DMA\_Offset = 0x1000 + (32 x n) + 0x0C

Access: Read/Write

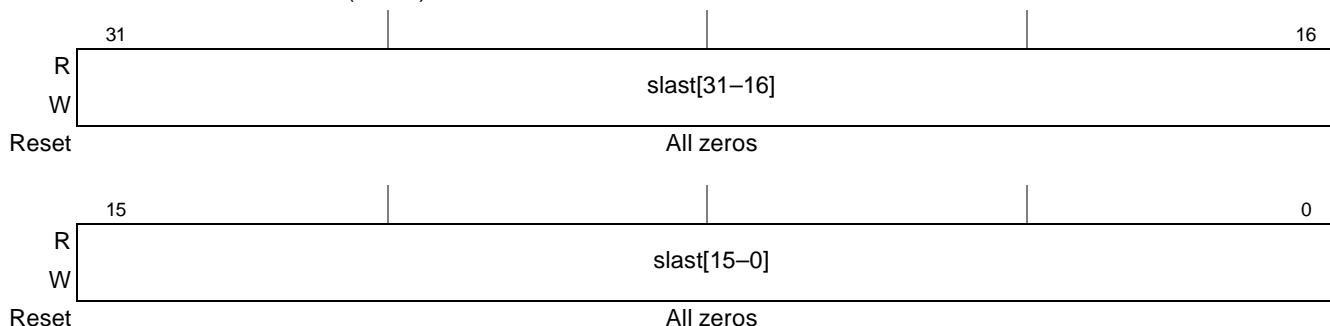


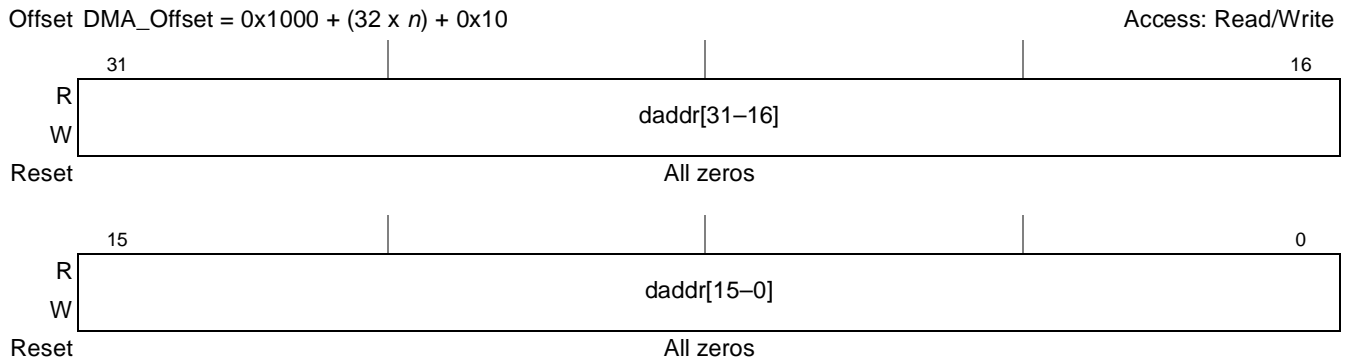
Figure 12-18. TCD Word 3 (TCDn.slast) Field

Table 12-19 describes the TCD word 3 fields.

Table 12-19. TCD Word 3 (TCD.slast) Field Descriptions

Bits	Name	Description
31-0	slast	Last source address adjustment. Adjustment value added to the source address at the completion of the outer major iteration count. This value can be applied to 'restore' the source address to the initial value, or adjust the address to reference the next data structure.

Figure 12-19 shows the TCD word 4 field.



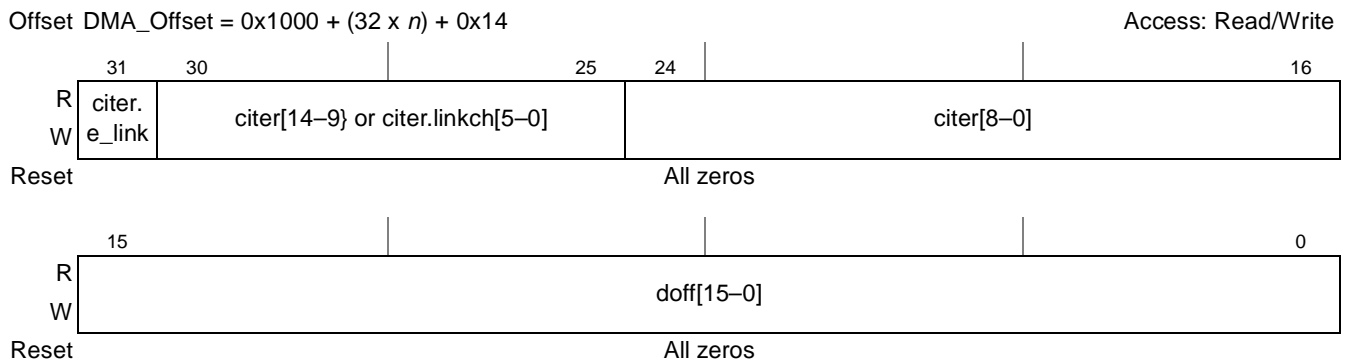
**Figure 12-19. TCD Word 4 (TCDn.daddr) Field**

Table 12-20 describes the TCD word 4 fields.

**Table 12-20. TCD Word 4 (TCD.daddr) Field Description**

Bits	Name	Description
31-0	daddr	Destination address. Memory address pointing to the destination data.

Figure 12-20 shows the TCD word 5 field.



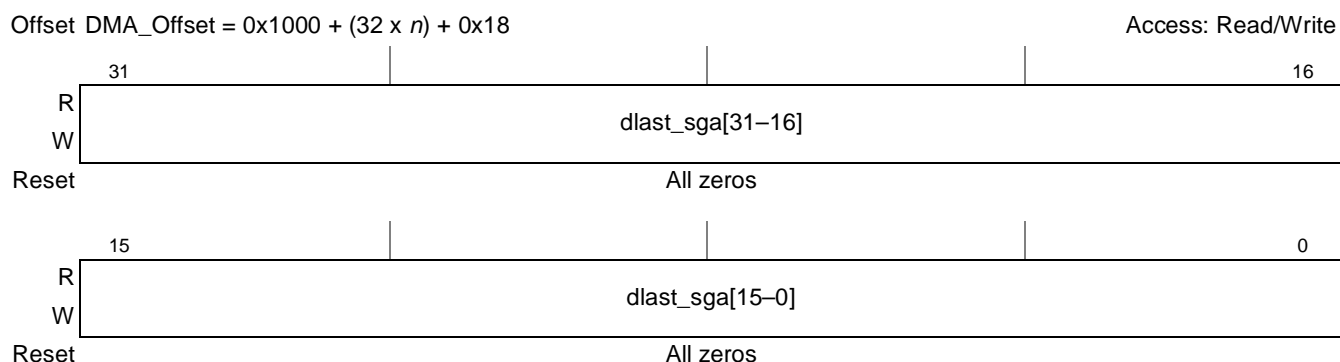
**Figure 12-20. TCD Word 5 (TCDn.{citer, doff}) Fields**

Table 12-21 describes the TCD word 5 fields.

**Table 12-21. TCD Word 5 (TCD.{citer, doff} Field Descriptions**

Bits	Name	Description
31	citer.e_link	Enable channel-to-channel linking on minor loop complete. As the channel completes the inner minor loop, this flag enables the linking to another channel, defined by citer.linkch[5:0]. The link target channel initiates a channel service request via an internal mechanism that sets the TCD.start bit of the specified channel. If channel linking is disabled, the citer value is extended to 15 bits in place of a link channel number. If the major loop is exhausted, this link mechanism is suppressed in favor of the major.e_link channel linking. This bit must be equal to the biter.e_link bit otherwise a configuration error is reported. 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled.
30–25	citer[14–9] or citer.linkch[5–0]	Current major iteration count or link channel number. If (TCD.citer.e_link = 0) then no channel-to-channel linking (or chaining) is performed after the inner minor loop is exhausted. TCD word 5, bits [30:25] are used to form a 15 bit citer field. Otherwise, after the minor loop is exhausted, the DMA engine initiates a channel service request at the channel defined by citer.linkch[5:0] by setting that channel's TCD.start bit. The value contained in citer.linkch[5:0] must not exceed the number of implemented channels.
24–16	citer[8–0]	Current major iteration count. This 9 or 15-bit count represents the current major loop count for the channel. It is decremented each time the minor loop is completed and updated in the transfer control descriptor memory. Once the major iteration count is exhausted, the channel performs a number of operations (e.g., final source and destination address calculations), optionally generating an interrupt to signal channel completion before reloading the citer field from the beginning iteration count (biter) field. When the citer field is initially loaded by software, it must be set to the same value as that contained in the biter field. If the channel is configured to execute a single service request, the initial values of biter and citer should be 0x0001.
15–0	doff[15–0]	Destination address signed offset. Sign-extended offset applied to the current destination address to form the next-state value as each destination write is completed.

Figure 12-21 shows the TCD word 6 field.



**Figure 12-21. TCD Word 6 (TCDn.dlast\_sga) Field**

Table 12-22 describes the TCD word 6 fields.

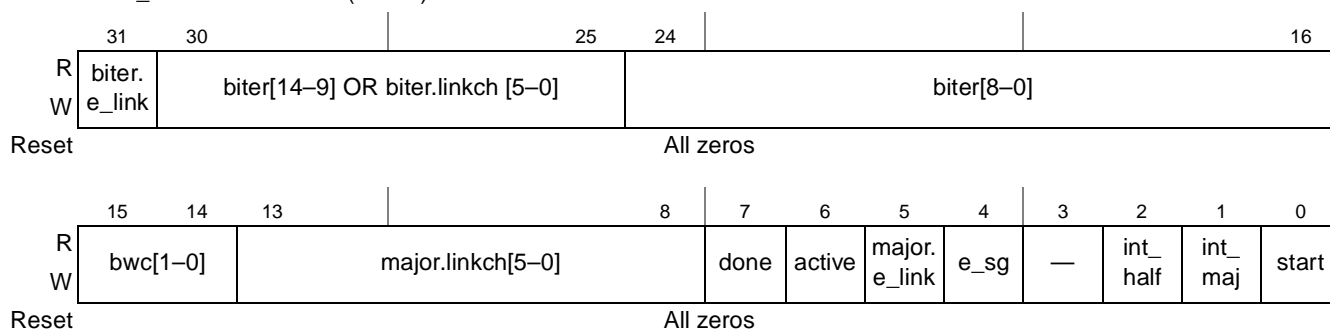
**Table 12-22. TCD Word 6 (TCD.dlast\_sga) Field Descriptions**

Bits	Name	Description
31–0	dlast_sga [31–0]	Last destination address adjustment or the memory address for the next transfer control descriptor to be loaded into this channel (scatter/gather). If (TCD.e_sg = 0), then Adjustment value added to the destination address at the completion of the outer major iteration count. This value can be applied to 'restore' the destination address to the initial value, or adjust the address to reference the next data structure. else, This address points to the beginning of a 0-modulo-32 region containing the next transfer control descriptor to be loaded into this channel. This channel reload is performed as the major iteration count completes. The scatter/gather address must be 0-modulo-32, else a configuration error is reported.

Figure 12-22 shows the TCD word 7 field.

Offset DMA\_Offset = 0x1000 + (32 x n) + 0x1C

Access: Read/Write



**Figure 12-22. TCD Word 7 (TCDn.{biter, control/status}) Fields**

Table 12-23 describes the TCD word 6 fields.

**Table 12-23. TCD Word 7 (TCD.{biter, control/status}) Field Descriptions**

Bits	Name	Description
31	biter.e_link	Enable channel-to-channel linking on minor loop complete. This is the initial value copied into the citer.e_link field when the major loop is completed. The citer.e_link field controls channel linking during channel execution. This bit must be equal to the citer.e_link bit otherwise a configuration error is reported. 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled.
30–25	biter[14–9] or biter.linkch [5–0]	Beginning major iteration count or beginning link channel number. This is the initial value copied into the citer field or citer.linkch field when the major loop is completed. The citer fields control the iteration count and linking during channel execution. If (TCD.biter.e_link = 0) then No channel-to-channel linking (or chaining) is performed after the inner minor loop is exhausted. TCD word 5, bits [30–25] are used to form a 15-bit biter field. Otherwise, after the minor loop is exhausted, the DMA engine initiates a channel service request at the channel defined by biter.linkch[5–0] by setting that channel's TCD.start bit. The value contained in biter.linkch[5–0] must not exceed the number of implemented channels.

Table 12-23. TCD Word 7 (TCD.{biter, control/status}) Field Descriptions (continued)

Bits	Name	Description
24–16	biter[8–0]	Beginning major iteration count. This is the initial value copied into the citer field or citer.linkch field when the major loop is completed. The citer fields controls the iteration count and linking during channel execution. This 9 or 15-bit counter presents the beginning major loop count for the channel. As the major iteration count is exhausted, the contents of the entire 16 bit biter entry is reloaded into the 16 bit citer entry. When the biter field is initially loaded by software, it must be set to the same value as that contained in the citer field. If the channel is configured to execute a single service request, the initial values of biter and citer should be 0x0001.
15–14	bwc[1–0]	Bandwidth control. This two-bit field provides a mechanism to effectively throttle the amount of bus bandwidth consumed by the DMA. In general, as the DMA processes the inner minor loop, it continuously generates read/write, read/write, ..., sequences until the minor count is exhausted. To minimize start-up latency, bandwidth control stalls are suppressed for the first two AHB bus cycles and after the last write of each minor loop. The dynamic priority elevation setting elevates the priority of the DMA as seen by the system arbiter for the executing channel. Dynamic priority elevation is suppressed during the first two AHB bus cycles. 00 No DMA engine stalls 01 dynamic priority elevation 10 DMA engine stalls for four cycles after each R/W 11 DMA engine stalls for eight cycles after each R/W
13–8	major.linkch [5–0]	Link channel number. If (TCD.major.e_link = 0) then No channel-to-channel linking (or chaining) is performed after the outer major loop counter is exhausted. Otherwise, after the major loop counter is exhausted, the DMA engine initiates a channel service request at the channel defined by major.linkch[5:0] by setting that channel's TCD.start bit. The value contained in major.linkch[5:0] must not exceed the number of implemented channels.
7	done	Channel done. This flag indicates the DMA has completed the outer major loop. It is set by the DMA engine as the citer count reaches zero; it is cleared by software, or the hardware when the channel is activated. This bit must be cleared in order to write the major.e_link or e_sg bits.
6	active	Channel active. This flag signals the channel is currently in execution. It is set when channel service begins, and is cleared by the DMA engine as the inner minor loop completes or if any error condition is detected.
5	major.e_link	Enable channel-to-channel linking on major loop complete. As the channel completes the outer major loop, this flag enables the linking to another channel, defined by major.linkch[5:0]. The link target channel initiates a channel service request via an internal mechanism that sets the TCD.start bit of the specified channel. To support the dynamic linking coherency model, this field is forced to zero when written to while the TCD.done bit is set. 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled.
4	e_sg	Enable scatter/gather processing. As the channel completes the outer major loop, this flag enables scatter/gather processing in the current channel. If enabled, the DMA engine uses dlast_sga as a memory pointer to a 0-modulo-32 address containing a 32-byte data structure which is loaded as the transfer control descriptor into the local memory. To support the dynamic scatter/gather coherency model, this field is forced to zero when written to while the TCD.done bit is set. 0 The current channel's TCD is "normal" format. 1 The current channel's TCD specifies a scatter gather format. The dlast_sga field provides a memory pointer to the next TCD to be loaded into this channel after the outer major loop completes its execution.
3	—	Reserved

Table 12-23. TCD Word 7 (TCD.{biter, control/status}) Field Descriptions (continued)

Bits	Name	Description
2	int_half	Enable an interrupt when major counter is half complete. If this flag is set, the channel generates an interrupt request by setting the appropriate bit in the DMAINT register when the current major iteration count reaches the halfway point. Specifically, the comparison performed by the DMA engine is $(citer == (biter \gg 1))$ . This halfway point interrupt request is provided to support double-buffered schemes or other types of data movement where the processor needs an early indication of the transfer's progress. The halfway complete interrupt is disabled when biter values are less than two. 0 The half-point interrupt is disabled 1 The half-point interrupt is enabled.
1	int_maj	Enable an interrupt when major iteration count completes. If this flag is set, the channel generates an interrupt request by setting the appropriate bit in the DMAINT register when the current major iteration count reaches zero. 0 The end-of-major loop interrupt is disabled. 1 The end-of-major loop interrupt is enabled.
0	start	Channel start. If this flag is set, the channel is requesting service. The DMA hardware automatically clears this flag after the channel begins execution. 0 The channel is not explicitly started. 1 The channel is explicitly started via a software initiated service request.

## 12.4 Functional Description

This section provides an overview of the microarchitecture and functional operation of the DMA module.

### 12.4.1 DMA Microarchitecture

The DMA module is partitioned into two major modules: the DMA engine and the transfer control descriptor local memory. Additionally, the DMA engine is further partitioned into four submodules, which are detailed below.

- DMA engine
  - *addr\_path*: This module implements registered versions of two channel transfer control descriptors: channel x and channel y, and is responsible for all the master bus address calculations. All the implemented channels provide the exact same functionality. This hardware structure allows the data transfers associated with one channel to be preempted after the completion of a read/write sequence if a higher priority channel service request is asserted while the first channel is active. Once a channel is activated, it runs until the minor loop is completed unless preempted by a higher priority channel. This capability provides a mechanism (optionally enabled by DCHPRIn[ECP]) where a large data move operation can be preempted to minimize the time another channel is blocked from execution. When any other channel is activated, the contents of its transfer control descriptor is read from the local memory and loaded into the registers of the other *addr\_path.channel\_{x,y}*. Once the inner minor loop completes execution, the *addr\_path* hardware writes the new values for the *TCD.{saddr, daddr, citer}* back into the local memory. If the major iteration count is exhausted, additional processing is performed, including the final address pointer updates, reloading the *TCD.citer* field, and a possible fetch of the next TCD from memory as part of a scatter/gather operation.



- *data\_path*: This module implements the actual bus master read/write data path. It includes 32 bytes of register storage (matching the maximum transfer size) and the necessary mux logic to support any required data alignment.
- *pmodel\_charb*: This module implements the first section of DMA programming model as well as the channel arbitration logic. The programming model registers are connected to the register interface (not shown). The `dma_ipi_int[n]` outputs are also connected to this module (via the control logic).
- *control*: This module provides all the control functions for the DMA engine. For data transfers where the source and destination sizes are equal, the DMA engine performs a series of source read, destination write operations until the number of bytes specified in the inner minor loop byte count has been moved. For descriptors where the sizes are not equal, multiple access of the smaller size data are required for each reference of the larger size. As an example, if the source size references 16-bit data and the destination is 32-bit data, two reads are performed, then one 32-bit write.
- *transfer\_control\_descriptor* local memory
  - *memory controller*: This logic implements the required dual-ported controller, handling accesses from both the DMA engine as well as references from the register interface. As noted earlier, in the event of simultaneous accesses, the DMA engine is given priority and the register interface transaction is stalled.
  - *memory array*: The TCD is implemented using a single-ported, synchronous compiled RAM memory array.

## 12.4.2 DMA Basic Data Flow

The basic flow of a data transfer can be partitioned into three segments. As shown in [Figure 12-23](#), the first segment involves the channel service request. Software requests the channel service by setting the `TCD.start` bit. In the next cycle, the channel arbitration is performed, either using the fixed-priority or round-robin algorithm. After the arbitration is complete, the activated channel number is sent through the address path (`addr_path`) and converted into the required address to access the TCD local memory. Next, the TCD memory is accessed and the required descriptor read from the local memory and loaded into the DMA engine `addr_path.channel_{x,y}` registers. The TCD memory is organized 64-bits in width to

minimize the time needed to fetch the activated channel's descriptor and load it into the DMA engine `addr_path.channel_{x,y}` registers.

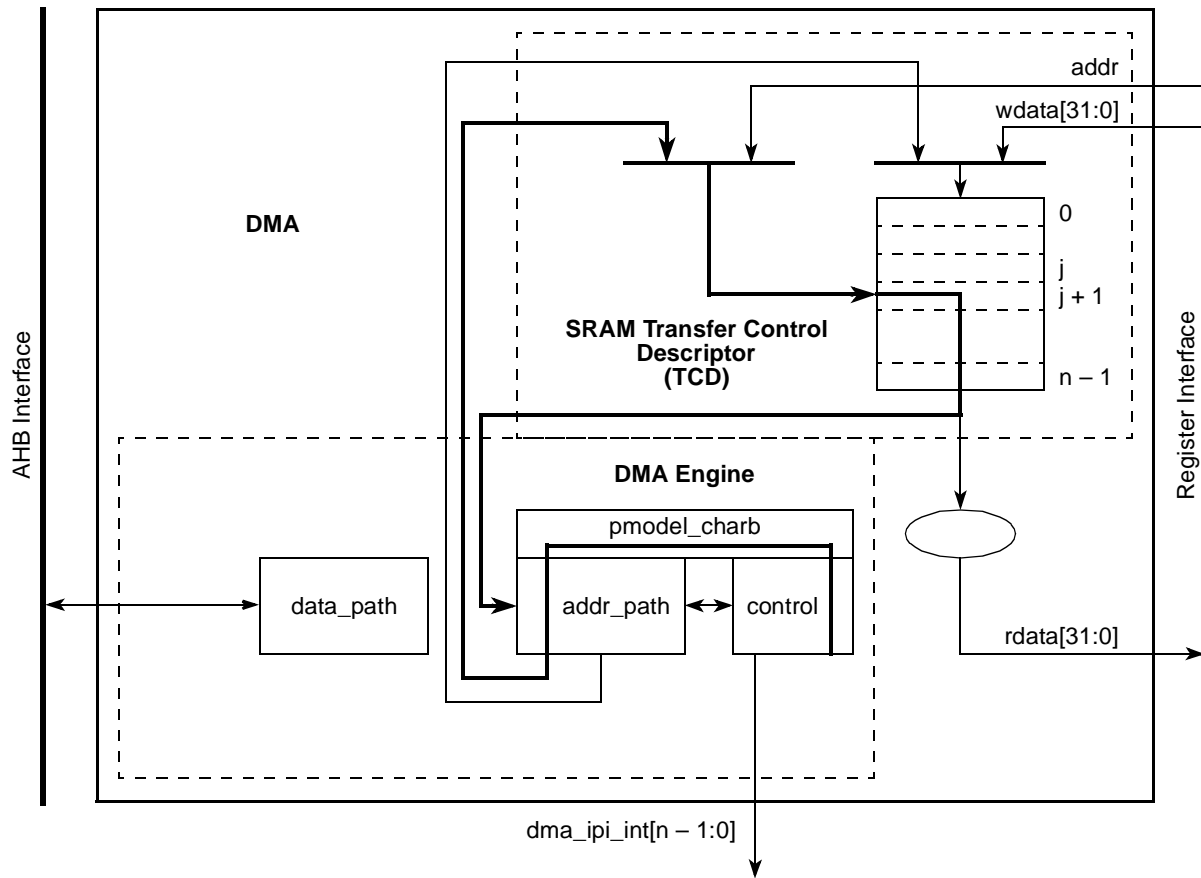
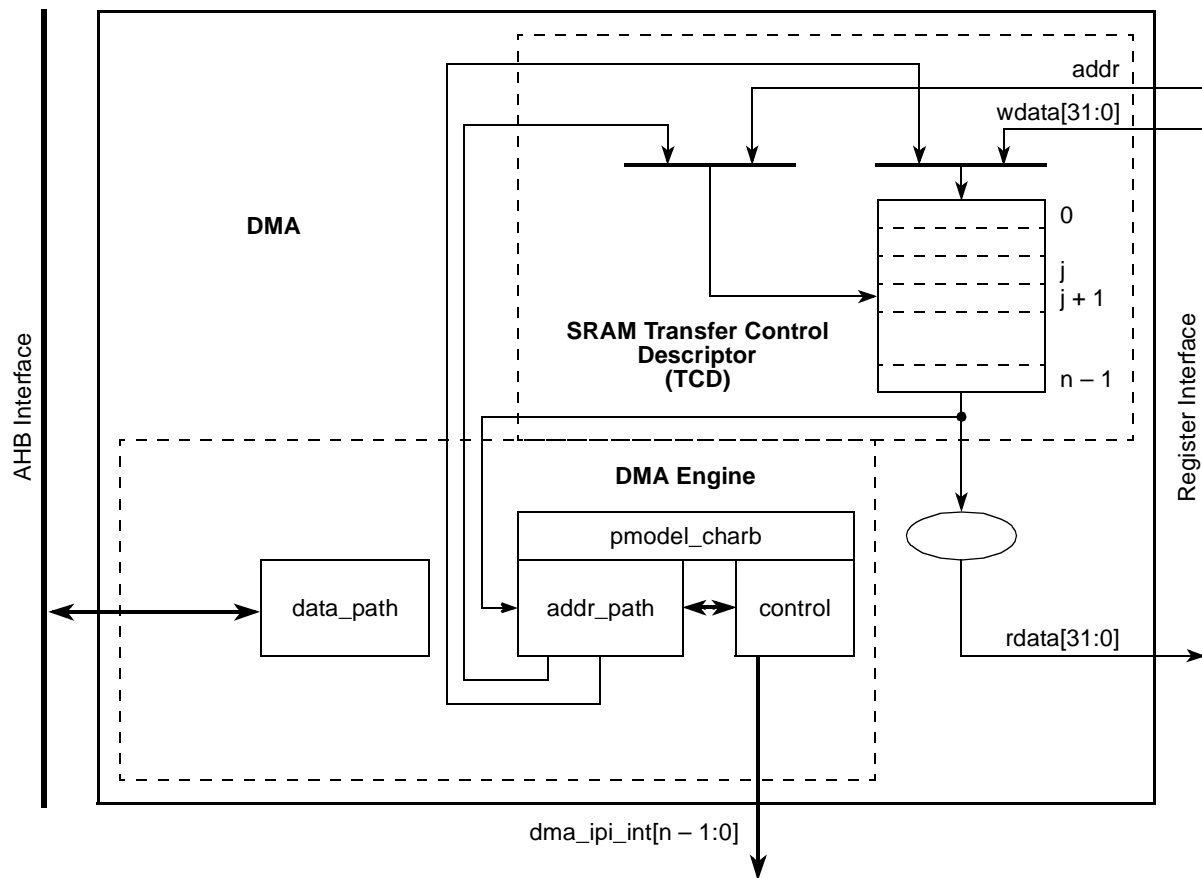


Figure 12-23. DMA Operation—Part 1

In the second part of the basic data flow as shown in Figure 12-24, the modules associated with the data transfer (`addr_path`, `data_path` and `control`) sequence through the required source reads and destination writes to perform the actual data movement. The source reads are initiated and the fetched data is temporarily stored in the `data_path` module until it is gated onto the AHB bus during the destination write.

This source read/destination write processing continues until the inner minor byte count has been transferred.



**Figure 12-24. DMA Operation—Part 2**

Once the inner minor byte count has been moved, the final phase of the basic data flow is performed. In this segment, the `addr_path` logic performs the required updates to certain fields in the channel's TCD, for example, `saddr`, `daddr`, `citer`. If the outer major iteration count is exhausted, then there are additional operations which are performed. These include the final address adjustments and reloading of the biter field into the `citer`. Additionally, assertion of an optional interrupt request occurs at this time, as does a possible fetch of a new TCD from memory using the scatter/gather address pointer included in the

descriptor. The updates to the TCD memory and the assertion of an interrupt request are shown in Figure 12-25.

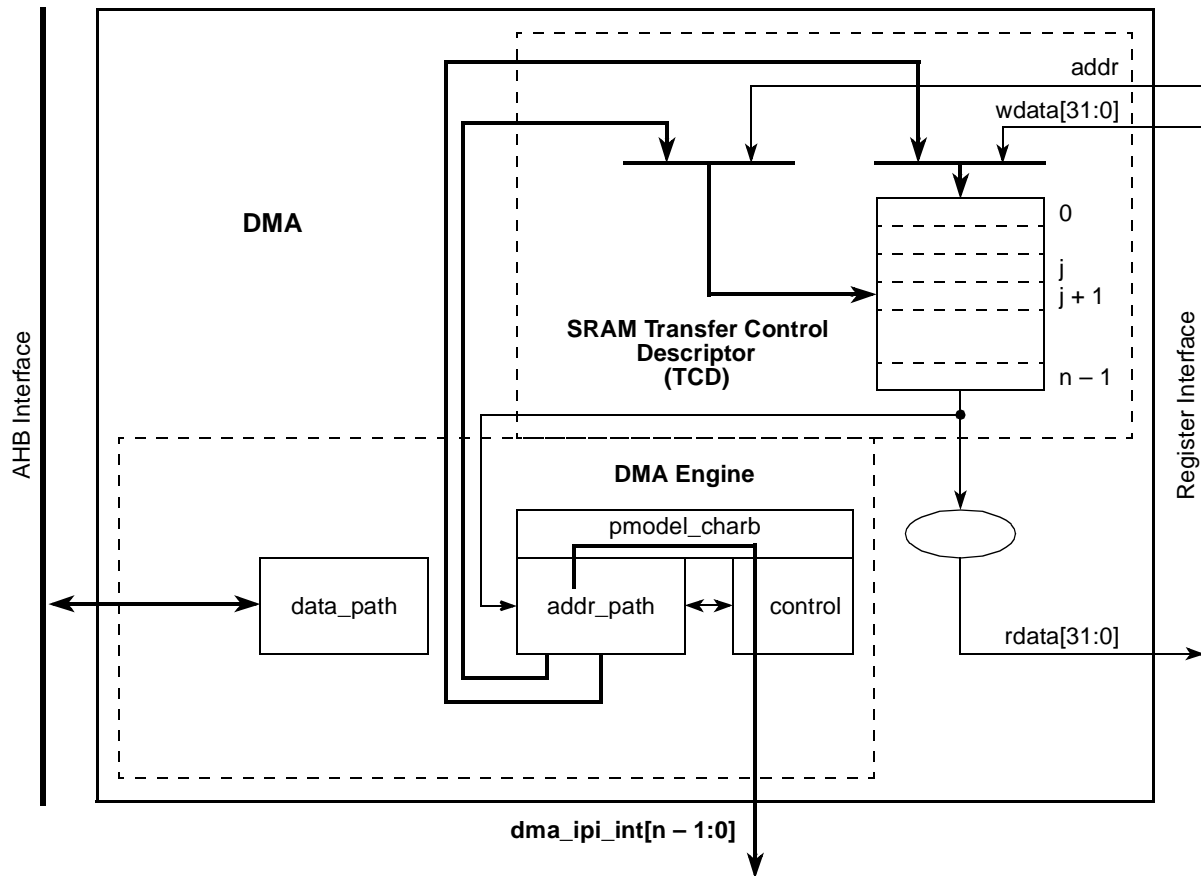


Figure 12-25. DMA Operation—Part 3

## 12.5 Initialization/Application Information

This section discusses the DMA initialization and programming errors.

### 12.5.1 DMA Initialization

The following sequence is a typical initialization of the DMA.

1. Write the DMACR register if a configuration other than the default is desired.
2. Write the channel priority levels into the DCHPRIn registers if a configuration other than the default is desired.
3. Enable error interrupts in the DMAEEI registers if so desired.
4. Write the 32 bytes TCD for each channel that may request service.
5. Request channel service by software (setting the `TCD.start` bit).

Once any channel requests service, a channel is selected for execution based on the arbitration and priority levels written into the programmer's model. The DMA engine reads the entire TCD for the selected

channel into its internal address path module. As the TCD is being read, the first transfer is initiated on the AHB bus unless a configuration error is detected. Transfers from the source (as defined by the source address, `TCD.saddr`) to the destination (as defined by the destination address, `TCD.daddr`) continue until the specified number of bytes (`TCD.nbytes`) have been transferred. When the transfer is complete, the DMA engine's local `TCD.saddr`, `TCD.daddr`, and `TCD.citer` are written back to the main TCD memory and any minor loop channel linking is performed, if enabled. If the major loop is exhausted, further post processing is executed, that is, interrupts, major loop channel linking, and scatter/gather operations, if enabled.

## 12.5.2 DMA Programming Errors

The DMA performs various tests on the transfer control descriptor to verify consistency in the descriptor data. Most programming errors are reported on a per channel basis with the exception of channel priority error and CPE in the DMAES register.

For all error types other than channel priority errors, the channel number causing the error is recorded in the DMAES register. If the error source is not removed before the next activation of the problem channel, the error is detected and recorded again.

In general, if priority levels are not unique, the highest channel priority that has an active request is selected, but the lowest numbered channel with that priority is selected by arbitration and executed by the DMA engine. The error interrupts and error reporting is associated with the selected channel.

## 12.6 DMA Transfer

This section discusses the procedures for single and multiple requests.

### 12.6.1 Single Request

To perform a simple transfer of  $n$  bytes of data with one activation, set the major loop to one (`TCD.citer = TCD.biter = 1`). The data transfer begins after the channel service request is acknowledged and the channel is selected to execute. Once the transfer is complete, the `TCD.done` bit is set and an interrupt is generated, if properly enabled.

For example, the following TCD entry is configured to transfer 16 bytes of data. The DMA is programmed for one iteration of the major loop transferring 16 bytes per iteration. The source memory has a byte wide memory port located at 0x1000. The destination memory has a word wide port located at 0x2000. The address offsets are programmed in increments to match the size of the transfer; one byte for the source and four bytes for the destination.

The final source and destination addresses are adjusted to return to their beginning values:

- `TCD.citer = TCD.biter = 1`
- `TCD.nbytes = 16`
- `TCD.saddr = 0x1000`
- `TCD.soff = 1`
- `TCD.ssize = 0`
- `TCD.slant = -16`

- `TCD.daddr = 0x2000`
- `TCD.doff = 4`
- `TCD.dsize = 2`
- `TCD.dlast_sga = -16`
- `TCD.int_maj = 1`
- `TCD.start = 1` (`TCD.word7` should be written last after all other fields have been initialized)
- All other TCD fields = 0

This would generate the following sequence of events:

1. Register interface write to the `TCD.start` bit requests channel service.
2. Software sets the `TCD.start` bit of the channel for activation. The channel is selected by arbitration for servicing.
3. DMA engine writes: `TCD.done = 0`, `TCD.start = 0`, `TCD.active = 1`.
4. DMA engine reads: channel TCD data from local memory to internal register file.
5. The source to destination transfers are executed as follows:
  - a) `read_byte(0x1000)`, `read_byte(0x1001)`, `read_byte(0x1002)`, `read_byte(0x1003)`
  - b) `write_word(0x2000)` -> first iteration of the minor loop
  - c) `read_byte(0x1004)`, `read_byte(0x1005)`, `read_byte(0x1006)`, `read_byte(0x1007)`
  - d) `write_word(0x2004)` -> second iteration of the minor loop
  - e) `read_byte(0x1008)`, `read_byte(0x1009)`, `read_byte(0x100a)`, `read_byte(0x100b)`
  - f) `write_word(0x2008)` -> third iteration of the minor loop
  - g) `read_byte(0x100c)`, `read_byte(0x100d)`, `read_byte(0x100e)`, `read_byte(0x100f)`
  - h) `write_word(0x200c)` -> last iteration of the minor loop -> major loop complete
6. DMA engine writes: `TCD.saddr = 0x1000`, `TCD.daddr = 0x2000`, `TCD.citer = 1` (`TCD.biter`).
7. DMA engine writes: `TCD.active = 0`, `TCD.done = 1`, `DMAINT[n] = 1`.
8. The channel retires.

The DMA goes idle or services next channel.

## 12.6.2 Multiple Requests

The next example is the same as previous with the exception of transferring 32 bytes by software triggered operation. The only fields that change are the major loop iteration count and the final address offsets. The DMA is programmed for two iterations of the major loop transferring 16 bytes per iteration.

- `TCD.citer = TCD.biter = 2`
- `TCD.slast = -32`
- `TCD.dlast_sga = -32`

This generates the following sequence of events:

1. Software sets the `TCD.start` bit of the channel for activation. The channel is selected by arbitration for servicing.

2. DMA engine writes: `TCD.done = 0`, `TCD.start = 0`, `TCD.active = 1`.
3. DMA engine reads: channel TCD data from local memory to internal register file.
4. The source to destination transfers are executed as follows:
  - a) `read_byte(0x1000)`, `read_byte(0x1001)`, `read_byte(0x1002)`, `read_byte(0x1003)`
  - b) `write_word(0x2000)` → first iteration of the minor loop
  - c) `read_byte(0x1004)`, `read_byte(0x1005)`, `read_byte(0x1006)`, `read_byte(0x1007)`
  - d) `write_word(0x2004)` → second iteration of the minor loop
  - e) `read_byte(0x1008)`, `read_byte(0x1009)`, `read_byte(0x100a)`, `read_byte(0x100b)`
  - f) `write_word(0x2008)` → third iteration of the minor loop
  - g) `read_byte(0x100c)`, `read_byte(0x100d)`, `read_byte(0x100e)`, `read_byte(0x100f)`
  - h) `write_word(0x200c)` → last iteration of the minor loop
5. DMA engine writes: `TCD.saddr = 0x1010`, `TCD.daddr = 0x2010`, `TCD.citer = 1`.
6. DMA engine writes: `TCD.active = 0`.
7. The channel retires → one iteration of the major loop.  
The DMA goes idle or services next channel.
8. Software sets the `TCD.start` bit of the channel for activation. The channel is selected by arbitration for servicing.
9. DMA engine writes: `TCD.done = 0`, `TCD.start = 0`, `TCD.active = 1`.
10. DMA engine reads: channel TCD data from local memory to internal register file.
11. The source to destination transfers are executed as follows:
  - a) `read_byte(0x1010)`, `read_byte(0x1011)`, `read_byte(0x1012)`, `read_byte(0x1013)`
  - b) `write_word(0x2010)` → first iteration of the minor loop
  - c) `read_byte(0x1014)`, `read_byte(0x1015)`, `read_byte(0x1016)`, `read_byte(0x1017)`
  - d) `write_word(0x2014)` → second iteration of the minor loop
  - e) `read_byte(0x1018)`, `read_byte(0x1019)`, `read_byte(0x101a)`, `read_byte(0x101b)`
  - f) `write_word(0x2018)` → third iteration of the minor loop
  - g) `read_byte(0x101c)`, `read_byte(0x101d)`, `read_byte(0x101e)`, `read_byte(0x101f)`
  - h) `write_word(0x201c)` → last iteration of the minor loop → major loop complete
12. DMA engine writes: `TCD.saddr = 0x1000`, `TCD.daddr = 0x2000`, `TCD.citer = 2` (`TCD.biter`).
13. DMA engine writes: `TCD.active = 0`, `TCD.done = 1`, `DMAINT[n] = 1`.
14. The channel retires → major loop complete.

The DMA goes idle or services the next channel.

## 12.7 TCD Status

This section discusses the two methods to test for minor loop completion and explains active channel TCD reads.

### 12.7.1 Minor Loop Complete

There are two methods to test for minor loop completion when using software initiated service requests. The first method is to read the `TCD.citer` field and test for a change. Another method may be extracted from the sequence shown below. The second method is to test the `TCD.start` bit AND the `TCD.active` bit. The minor loop complete condition is indicated by both bits reading zero after the `TCD.start` was written to a one. Polling the `TCD.active` bit may be inconclusive because the active status may be missed if the channel execution is short in duration. The TCD status bits execute the following sequence for a software activated channel:

1. `TCD.start = 1, TCD.active = 0, TCD.done = 0` (channel service request via software)
2. `TCD.start = 0, TCD.active = 1, TCD.done = 0` (channel is executing)
3. `TCD.start = 0, TCD.active = 0, TCD.done = 0` (channel has completed the minor loop and is idle) Or
4. `TCD.start = 0, TCD.active = 0, TCD.done = 1` (channel has completed the major loop and is idle)

The major loop complete status is explicitly indicated through the `TCD.done` bit.

The `TCD.start` bit is cleared automatically when the channel begins execution.

### 12.7.2 Active Channel TCD Reads

While the channel is executing, if the `TCD.saddr`, `TCD.daddr`, and `TCD.nbytes` are read, the true values of these fields are returned. The true values of `TCD.saddr`, `TCD.daddr`, and `TCD.nbytes` are the values that the DMA engine is currently using in its internal register file (true values are not the values in the TCD local memory for that channel).

The addresses (`saddr` and `daddr`) and `nbytes` (decrements to zero as the transfer progresses) can give an indication of the progress of the transfer. All other values are read back from the TCD local memory.

### 12.7.3 Preemption status

Preemption is only available when fixed arbitration is selected for the channel arbitration modes. A preempt-able situation is one in which a preempt-enabled channel is running and a higher priority request becomes active. When the DMA engine is not operating in the fixed channel arbitration mode, the determination of the relative priority of the actively running and the outstanding requests become undefined. Channel priorities are treated as equal (or more exactly, constantly rotating) when round-robin arbitration mode is selected.

The `TCD.active` bit for the preempted channel remains asserted throughout the preemption. The preempted channel is temporarily suspended while the preempting channel executes one iteration of the major loop. Two `TCD.active` bits set at the same time in the overall TCD map indicates a higher priority channel is actively preempting a lower priority channel.



## 12.8 Channel Linking

Channel linking (or chaining) is a mechanism where one channel sets the `TCD.start` bit of another channel (or itself), and initiates a service request for that channel. This operation is automatically performed by the DMA engine at the conclusion of the major or minor loop when properly enabled.

The minor loop channel linking occurs at the completion of the minor loop (or one iteration of the major loop). The `TCD.citer.e_link` field determines if a minor loop link is requested. When enabled, the channel link is made after each iteration of the major loop except for the last. When the major loop is exhausted, only the major loop channel link fields are used to determine if a channel link should be made. For example, with the initial fields of:

- `TCD.citer.e_link = 1`
- `TCD.citer.linkch = 0xC`
- `TCD.citer value = 0x4`
- `TCD.major.e_link = 1`
- `TCD.major.linkch = 0x7`

With the bits set as mentioned above, the DMA operation executes as:

1. minor loop done -> set channel 12 `TCD.start` bit
2. minor loop done -> set channel 12 `TCD.start` bit
3. minor loop done -> set channel 12 `TCD.start` bit
4. minor loop done, major loop done -> set channel 7 `TCD.start` bit

When minor loop linking is enabled (`TCD.citer.e_link = 1`), the `TCD.citer` field uses a nine bit vector to form the current iteration count. When minor loop linking is disabled (`TCD.citer.e_link = 0`), the `TCD.citer` field uses a 15 bit vector to form the current iteration count. The bits associated with the `TCD.citer.linkch` field are concatenated onto the `citer` value to increase the range of the `citer`.

### NOTE

The `TCD.citer.e_link` bit and the `TCD.biter.e_link` bit must equal or a configuration error is reported. The `citer` and `biter` vector widths must be equal in order to calculate the major loop, half-way done interrupt point.

## 12.9 Programming during channel execution

This section provides recommended methods to change the programming model during channel execution.

### 12.9.1 Dynamic priority changing

The following options are recommended for dynamically changing channel priority levels:

- Switch to round-robin channel arbitration mode, change the channel priorities, and then switch back to fixed arbitration mode.
- Disable all the channels, change the channel priorities, and then enable the appropriate channels.

## 12.9.2 Dynamic channel linking and dynamic scatter/gather

Dynamic channel linking and dynamic scatter/gather is the process of changing the `TCD.major.e_link` or `TCD.e_sg` bits during channel execution. These bits are read from the TCD local memory at the end of channel execution. Therefore, allows the user to enable either feature during channel execution.

Because the user is allowed to change the configuration during execution, a coherency model is needed. Consider the scenario where the user attempts to execute a dynamic channel link by enabling the `TCD.major.e_link` bit at the same time the DMA engine is retiring the channel. The `TCD.major.e_link` would be set in the programmer's model, but it would be unclear whether the actual link was made before the channel retired.

The following coherency model is recommended when executing a dynamic channel link or dynamic scatter/gather request:

1. Set the `TCD.major.e_link` bit.
2. Read back the `TCD.major.e_link` bit.
3. Test the `TCD.major.e_link` request status:
  - a) If the bit is set, the dynamic link attempt is successful.
  - b) If the bit is cleared, the attempted dynamic link did not succeed, as the channel was already retiring.

This same coherency model is true for dynamic scatter/gather operations. For both dynamic requests, the TCD local memory controller forces the `TCD.major.e_link` and `TCD.e_sg` bits to zero on any writes to a channel's `TCD.word7` once that channel's `TCD.done` bit is set indicating the major loop is complete.

### NOTE

The user must clear the `TCD.done` bit before writing the `TCD.major.e_link` or `TCD.e_sg` bits. The `TCD.done` bit is cleared automatically by the DMA engine once a channel begins execution.

## Chapter 13

# Universal Serial Bus Interface

This chapter describes the universal serial bus (USB) interface 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*
- *On-The-Go Supplement to the USB 2.0 Specification, Revision 1.0a*

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*

### 13.1 Introduction

The device implements a dual-role (DR) USB module. This module may be connected to an external port. Collectively the module and external port are called the USB interface. The USB interface is shown in [Figure 13-1](#).

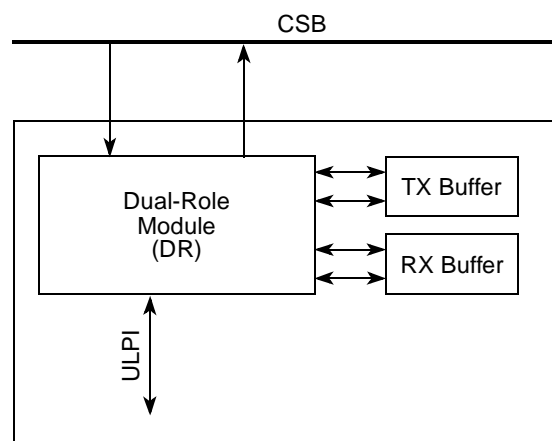


Figure 13-1. USB Interface Block Diagram

### 13.1.1 Overview

The USB DR module is a USB 2.0-compliant serial interface engine for implementing a USB interface. The registers and data structures for the module are based on the *Enhanced Host Controller Interface Specification for Universal Serial Bus* (EHCI) from Intel Corporation. The DR module can act as a device or host controller. Interfaces to negotiate the host or device role on the bus in compliance with the On-The-Go (OTG) supplement to the USB specification are also provided.

The DR module supports the required signaling for UTMI low pin count interface (ULPI) transceivers (PHYs). An external PHY would be used to interface to ULPI.

The 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.

### 13.1.2 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 three programmable, bidirectional USB endpoints
- Host and device support
- OTG (on-the-go) support, which includes both device and host capability, with external PHY (ULPI)

### 13.1.3 Modes of Operation

The USB DR module has three basic operating modes: host, device, and OTG.

#### NOTE

Only high-speed and full-speed operations are supported in device mode.

## 13.2 External Signals

This section contains detailed descriptions of all the USB dual-role controller signals. Many of the signals for the PHY interfaces are muxed onto the same pins in order to reduce pin count. Table 13-1 describes the signals, indicating which interface supports each signal.

**Table 13-1. USB External Signals**

Signal	I/O
USBDR_PWR_FAULT	I
USBDR_CLK	I
USBDR_DIR	I
USBDR_NXT	I
USBDR_TXDRXD[0:7]	I/O
USBDR_PCTL[0:1]	O
USBDR_STP	O

### 13.2.1 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 13-2 describes the signals for the ULPI interface.

**Table 13-2. ULPI Signal Descriptions**

Signal	I/O	Description	
USBDR_DIR	I	Direction. USBDR_DIR controls the direction of the data bus. When the PHY has data to transfer to USB port, it drives USBDR_DIR high to take ownership of the bus. When the PHY has no data to transfer it drives USBDR_DIR low and monitors the bus for link activity. The PHY pulls USBDR_DIR high whenever the interface cannot accept data from the link.	
		<b>State Meaning</b>	Asserted—PHY has data to transfer to the link. Negated—PHY has no data to transfer.
		<b>Timing</b>	Synchronous to PHY_CLK.
USBDR_NXT	I	Next data. The PHY asserts USBDR_NXT to throttle the data. When USB port is sending data to the PHY, USBDR_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, USBDR_NXT indicates when a new byte is available for USB port to consume.	
		<b>State Meaning</b>	Asserted—PHY is ready to transfer byte. Negated—PHY is not ready.
		<b>Timing</b>	Synchronous to PHY_CLK.

Table 13-2. ULPI Signal Descriptions (continued)

Signal	I/O	Description
USBDR_STP	O	Stop. USBDR_STP indicates the end of a transfer on the bus.
		<b>State Meaning</b> 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, USBDR_STP indicates the last byte of data was previously on the bus. If the PHY is sending data to USB port, USBDR_STP forces the PHY to end its transfer, negate USBDR_DIR and relinquish control of the data bus to the USB port. Negated—Indicates normal operation.
		<b>Timing</b> Synchronous to PHY_CLK.
USBDR_PWR_FAULT	I	Power fault. USBDR_PWR_FAULT indicates whether a power fault occurred on the USB port Vbus.
		<b>State Meaning</b> Asserted—Indicates that a Vbus fault occurred. Applications that support power switching must shut down Vbus power. Negated—Indicates normal operation.
		<b>Timing</b> Synchronous to PHY_CLK.
USBDR_PCTL0	O	Port control 0. USBDR_PCTL0 controls the port status indicator LED 0 when in host mode.
		<b>State Meaning</b> Asserted—LED on. Negated—LED off.
		<b>Timing</b> Synchronous to PHY_CLK.
USBDR_PCTL1	O	Port control 1. USBDR_PCTL1 controls the port status indicator LED 1 when in host mode.
		<b>State Meaning</b> Asserted—LED on. Negated—LED off.
		<b>Timing</b> Synchronous to PHY_CLK.
USBDR_TXDRXD[0:7]	I/O	Data bit <i>n</i> . USBDR_TXDRXD <sub><i>n</i></sub> is bit <i>n</i> of the 8-bit (USBDR_TXDRXD7–USBDR_TXDRXD0), uni-directional data bus used to carry USB, register, and interrupt data between the PHY and the USB controller.
		<b>State Meaning</b> Asserted—Data bit <i>n</i> is 1. Negated—Data bit <i>n</i> is 0.
		<b>Timing</b> Synchronous to PHY_CLK.
USBDR_CLK	I	Clocking signal for ULPI PHY interface.

### 13.3 Memory Map/Register Definitions

This section provides the memory map and detailed descriptions of all USB interface registers. The memory map of the USB interface is shown in [Table 13-3](#).

Table 13-3. USB Interface Memory Map

Offset	Register	Access	Reset	Section/Page
<b>USB DR Controller Registers</b>				
<b>USB DR Controller—Block Base Address 0x2_3000</b>				
0x000–0x0FF	Reserved, should be cleared	—	—	—

Table 13-3. USB Interface Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
0x100	CAPLENGTH—Capability register length	R	0x40	13.3.1.1/13-6
0x102	HCIVERSION—Host interface version number <sup>1</sup>	R	0x0100	13.3.1.2/13-7
0x104	HCSPARAMS—Host controller structural parameters <sup>1</sup>	R	0x0001_0011	13.3.1.3/13-7
0x108	HCCPARAMS—Host controller capability parameters <sup>1</sup>	R	0x0000_0006	13.3.1.4/13-8
0x120	DCIVERSION—Device interface version number	R	0x0001	13.3.1.5/13-9
0x124	DCCPARAMS—Device controller parameters	R	0x0000_0183	13.3.1.6/13-10
0x140	USBCMD—USB command	Mixed	0x0008_0000	13.3.2.1/13-10
0x144	USBSTS—USB status	Mixed	0x0000_0000	13.3.2.2/13-13
0x148	USBINTR—USB interrupt enable	R/W	0x0000_0000	13.3.2.3/13-15
0x14C	FRINDEX—USB frame index	R/W	0x0000_0000	13.3.2.4/13-16
0x154	PERIODICLISTBASE—Frame list base address <sup>1</sup>	R/W	0x0000_0000	13.3.2.6/13-18
	DEVICEADDR—USB device address	R/W	0x0000_0000	13.3.2.7/13-18
0x158	ASYNCLISTADDR—Next asynchronous list addr (host mode) <sup>1</sup>	R/W	0x0000_0000	13.3.2.8/13-19
	ENDPOINTLISTADDR—Address at endpoint list (device mode)	R/W	0x0000_0000	13.3.2.9/13-20
0x160	BURSTSIZE—Programmable burst size	R/W	0x0000_1010	13.3.2.10/13-20
0x164	TXFILLTUNING—Host TT transmit pre-buffer packet tuning	R/W	0x0000_0000	13.3.2.11/13-21
0x170	ULPI VIEWPORT—ULPI Register Access	Mixed	0x0000_0000	13.3.2.12/13-23
0x180	CONFIGFLAG—Configured flag register	R	0x0000_0001	13.3.2.13/13-24
0x184	PORTSC—Port status/control	Mixed	0x1000_0000	13.3.2.14/13-25
0x1A4	OTGSC—On-The-Go status and control <sup>1</sup>	Mixed	0x202C_2000	13.3.2.15/13-30
0x1A8	USBMODE—USB device mode	R/W	0x0000_0000	13.3.2.16/13-32
0x1AC	ENDPTSETUPSTAT—Endpoint setup status	R/W	0x0000_0000	13.3.2.17/13-33
0x1B0	ENDPOINTPRIME—Endpoint initialization	R/W	0x0000_0000	13.3.2.18/13-34
0x1B4	ENDPTFLUSH—Endpoint de-initialize	R/W	0x0000_0000	13.3.2.19/13-35
0x1B8	ENDPTSTATUS—Endpoint status	R	0x0000_0000	13.3.2.20/13-35
0x1BC	ENDPTCOMPLETE—Endpoint complete	w1c	0x0000_0000	13.3.2.21/13-36
0x1C0	ENDPTCTRL0—Endpoint control 0	Mixed	0x0080_0080	13.3.2.22/13-37
0x1C4	ENDPTCTRL1—Endpoint control 1	R/W	0x0000_0000	13.3.2.23/13-38
0x1C8	ENDPTCTRL2—Endpoint control 2	R/W	0x0000_0000	13.3.2.23/13-38
0x1CA–0x1D4	Reserved	—	—	—
0x400	SNOOP1—Snoop 1	R/W	0x0000_0000	13.3.2.24/13-39
0x404	SNOOP2—Snoop 2	R/W	0x0000_0000	13.3.2.24/13-39
0x408	AGE_CNT_THRESH—Age count threshold	R/W	0x0000_0000	13.3.2.25/13-40

**Table 13-3. USB Interface Memory Map (continued)**

Offset	Register	Access	Reset	Section/Page
0x40C	PRI_CTRL—Priority control	R/W	0x0000_0000	<a href="#">13.3.2.26/13-42</a>
0x410	SI_CTRL—System interface control	R/W	0x0000_0000	<a href="#">13.3.2.27/13-42</a>
0x500	CONTROL—Control	R/W	0x0000_0000	<a href="#">13.3.2.28/13-43</a>
0x504– 0xFFFF	Reserved, should be cleared	—	—	—

<sup>1</sup> This register has separate functions for the host and device operation; the host function is listed first in the table.

The following sections provide details about the registers in the USB memory map.

**NOTE**

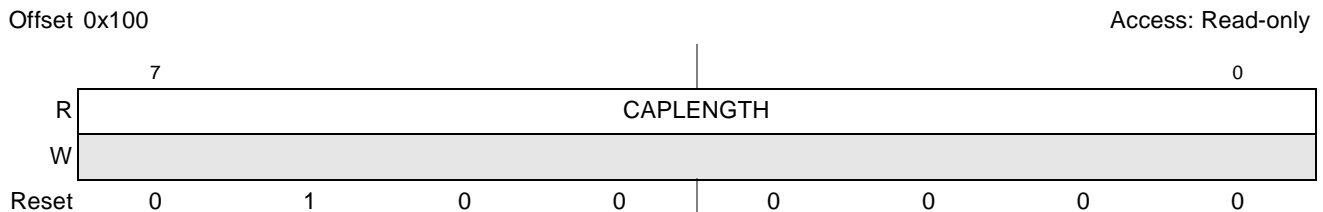
Memory may be viewed from either a big-endian or little-endian byte ordering perspective depending on the processor configuration. In big-endian mode, the most-significant byte of word 0 is located at address 0 and the least-significant byte of word 0 is located at address 3. In little-endian mode, the 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 DR 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.

**13.3.1 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.

**13.3.1.1 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 13-2](#) shows CAPLENGTH.



**Figure 13-2. Capability Registers Length (CAPLENGTH)**



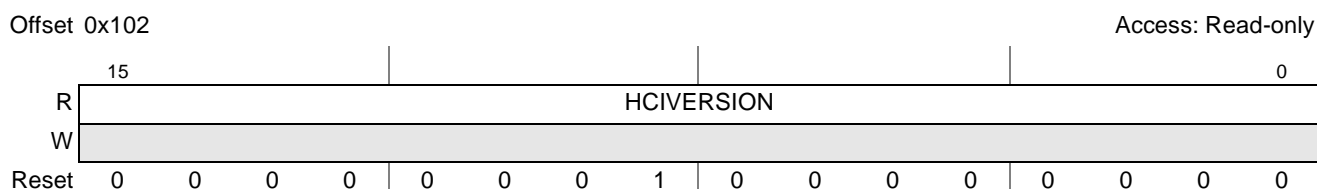
Table 13-4 provides bit descriptions for the CAPLENGTH register.

**Table 13-4. CAPLENGTH Register Field Descriptions**

Bits	Name	Description
7–0	CAPLENGTH	Capability registers length. Value is 0x40.

### 13.3.1.2 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 13-3 shows the HCIVERSION register.



**Figure 13-3. Host Controller Interface Version (HCIVERSION)**

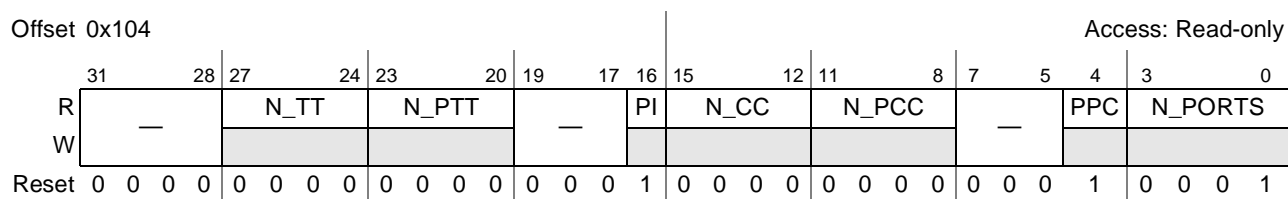
Table 13-5 provides bit descriptions for the HCIVERSION register.

**Table 13-5. HCIVERSION Register Field Descriptions**

Bits	Name	Description
15–0	—	EHCI revision number. Value is 0x0100 indicating version 1.0.

### 13.3.1.3 Host Controller Structural Parameters (HCSPARAMS)

HCSPARAMS contains structural parameters such as the number of downstream ports. Figure 13-4 shows the HCSPARAMS register.



**Figure 13-4. Host Controller Structural Parameters (HCSPARAMS)**

Table 13-6 provides bit descriptions for the HCSPARAMS register.

**Table 13-6. HCSPARAMS Register Field Descriptions**

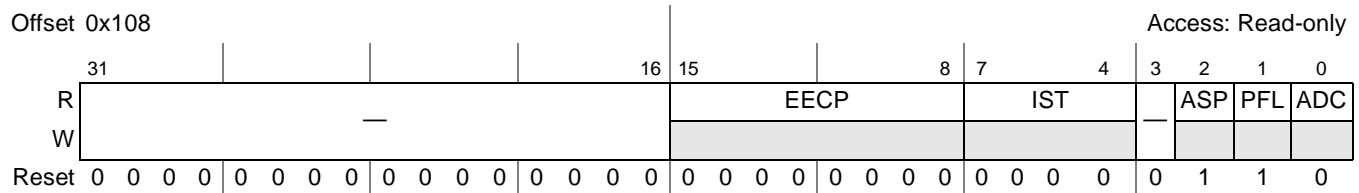
Bits	Name	Description
31–28	—	Reserved, should be cleared.
27–24	N_TT	Number of transaction translators. This is a non-EHCI field. This field indicates the number of embedded transaction translators associated the module. Always 1. See Section 13.9.1, “Embedded Transaction Translator Function.”

**Table 13-6. HCSPARAMS Register Field Descriptions (continued)**

Bits	Name	Description
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. 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 DR controller. Always 0.
11–8	N_PCC	Number ports per CC. This field indicates the number of ports supported per internal companion controller. Always 0.
7–5	—	Reserved, should be cleared.
4	PPC	Power port control. Indicates whether the host controller supports port power control. Always 1. 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 1.

### 13.3.1.4 Host Controller Capability Parameters (HCCPARAMS)

HCCPARAMS identifies multiple mode control (time-base bit functionality) addressing capability. Figure 13-5 shows the HCCPARAMS register.



**Figure 13-5. Host Control Capability Parameters (HCCPARAMS)**

Table 13-7 provides bit descriptions for the HCCPARAMS register.

**Table 13-7. 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. 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. This field is always 0.
3	—	Reserved, should be cleared.

**Table 13-7. HCCPARAMS Register Field Descriptions (continued)**

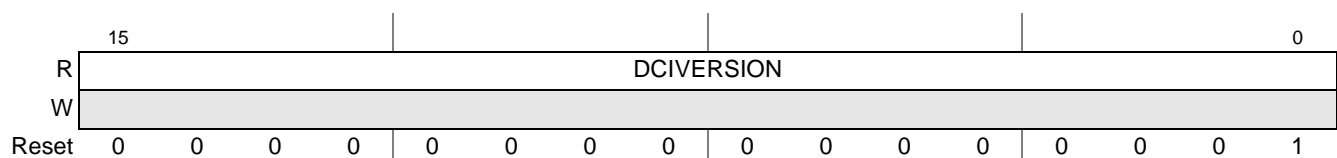
Bits	Name	Description
2	ASP	Asynchronous schedule park capability. Indicates whether the USB DR module supports the park feature for high-speed queue heads in the asynchronous schedule. The feature can be disabled or enabled and set to a specific level using the asynchronous schedule park mode enable and asynchronous schedule park mode count fields in the USBCMD register. 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 than 1024 elements. Frame list size is configured via the USBCMD register frame list size field. The frame list must always be aligned on a 4-K page boundary. This requirement ensures that the frame list is always physically contiguous. This field is always 1.
0	ADC	64-bit addressing capability. Always 0; 64-bit addressing is not supported. 0 Data structures use 32-bit address memory pointers

### 13.3.1.5 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 13-6](#) shows the DCIVERSION register.

Offset 0x120

Access: Read-only


**Figure 13-6. Device Interface Version (DCIVERSION)**

[Table 13-8](#) provides bit descriptions for the DCIVERSION register.

**Table 13-8. DCIVERSION Register Field Descriptions**

Bits	Name	Description
15-0	DCIVERSION	Device interface revision number.

### 13.3.1.6 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 DR module. Figure 13-7 shows the DCCPARAMS register.

Offset 0x124

Access: Read-only

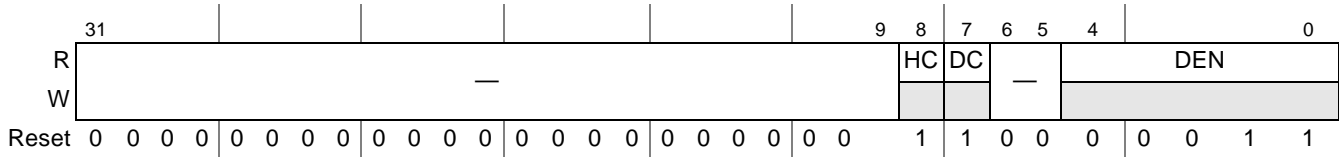


Figure 13-7. Device Control Capability Parameters (DCCPARAMS)

Table 13-9 provides bit descriptions for the DCCPARAMS register.

Table 13-9. DCCPARAMS Register Field Descriptions

Bits	Name	Description
31–9	—	Reserved, should be cleared.
8	HC	Host capable. Always 1, indicating the USB DR controller can operate as an EHCI compatible USB 2.0 host
7	DC	Device capable. Always 1, indicating the USB DR controller can operate as an USB 2.0 device. 1 Device capability. 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 0x3.

### 13.3.2 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.

#### 13.3.2.1 USB Command Register (USBCMD)

Figure 13-8 shows the USB command register. The module executes the command indicated in this register.

Offset 0x140

Access: Mixed

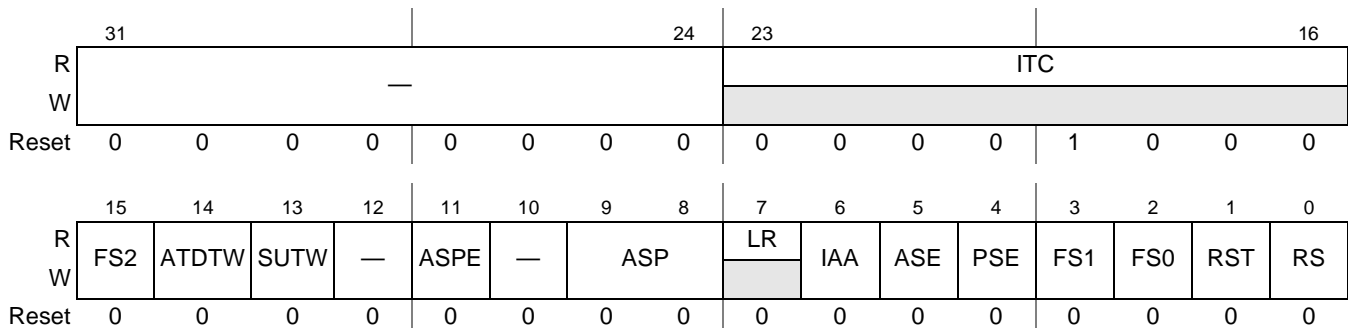


Figure 13-8. USB Command Register (USBCMD)

Table 13-10 provides bit descriptions for the USBCMD register.

**Table 13-10. 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 DR module issues interrupts. ITC contains the maximum interrupt interval measured in microframes. Valid values are shown below. 0x00 Immediate (no threshold) 0x01 1 microframe 0x02 2 microframes 0x04 4 microframes 0x08 8 microframes 0x10 16 microframes 0x20 32 microframes 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 its 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 <a href="#">Section 13.9.2, “Device Operation.”</a>
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 <a href="#">Section 13.9.2, “Device Operation.”</a>
12	—	Reserved, should be cleared.
11	ASPE	Asynchronous schedule park mode enable. Software uses this bit to enable or disable park mode. 0 Disabled 1 Enabled
10	—	Reserved, should be cleared.
9–8	ASP	Asynchronous schedule park mode count. This field defaults to 0x3 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 results 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 DR controller to issue an interrupt the next time it advances asynchronous schedule. Software must write a 1 to this bit to ring the doorbell. When the controller has evicted all appropriate cached schedule states, it sets USBSTS[AAI]. If USBINTR[AAE] is set, the host controller asserts an interrupt at the next interrupt threshold. 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 yields undefined results. This bit is only used in host mode. Setting this bit when the USB DR module is in device mode is selected results in undefined results.

**Table 13-10. USBCMD Register Field Descriptions (continued)**

Bits	Name	Description
5	ASE	Asynchronous schedule enable. Controls whether the controller skips processing the asynchronous schedule. Only used in host mode. 0 Do not process the asynchronous schedule 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. 0 Do not process the periodic schedule. 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. 000 1024 elements (4096 bytes) 001 512 elements (2048 bytes) 010 256 elements (1024 bytes) 011 128 elements (512 bytes) 100 64 elements (256 bytes) 101 32 elements (128 bytes) 110 16 elements (64 bytes) 111 8 elements (32 bytes)
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. Host mode: <ul style="list-style-type: none"> <li>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 results in undefined behavior.</li> </ul> Device mode: <ul style="list-style-type: none"> <li>When software sets this bit, the USB DR 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.</li> </ul>
0	RS	Run/Stop. Host mode: <ul style="list-style-type: none"> <li>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 DR 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).</li> </ul> Device mode: <ul style="list-style-type: none"> <li>Setting this bit causes the USB DR 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 is 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 causes a detach event.</li> </ul> 0 Stop 1 Run

### 13.3.2.2 USB Status Register (USBSTS)

Figure 13-9 shows the USB status register, which indicates various states of the USB DR 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).

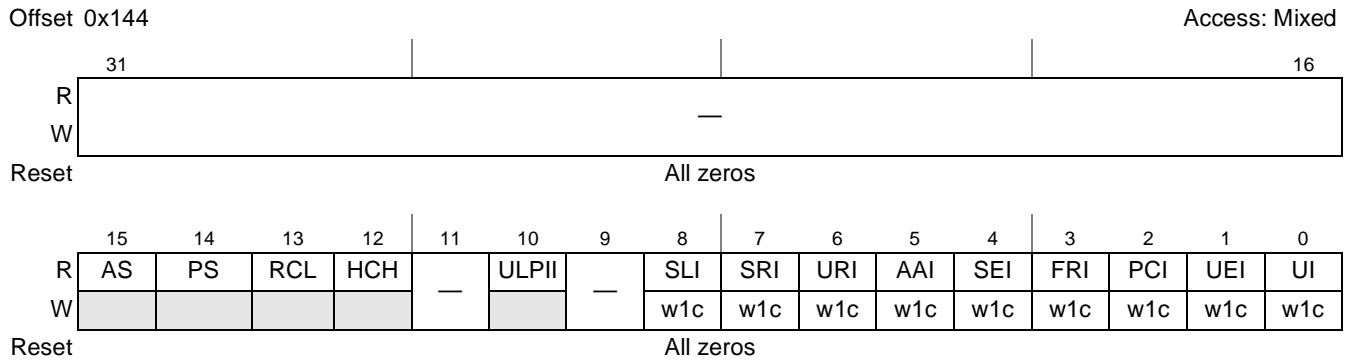


Figure 13-9. USB Status Register (USBSTS)

Table 13-11 shows the USBSTS register field descriptions.

Table 13-11. 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 DR controller is not required to immediately disable or enable the asynchronous schedule when software transitions USB_CMD[ASE]. When this bit and USB_CMD[ASE] have the same value, the asynchronous schedule is either enabled (1) or disabled (0). Only used in host mode. 0 Disabled 1 Enabled
14	PS	Periodic schedule status. Reports the current real status of the periodic schedule. The USB DR controller is not required to immediately disable or enable the periodic schedule when software transitions USB_CMD[PSE]. When this bit and USB_CMD[PSE] have the same value, the periodic schedule is either enabled (1) or disabled (0). Only used in host mode. 0 Disabled 1 Enabled
13	RCL	Reclamation. Used to detect an empty asynchronous schedule. Only used by the host mode. 0 Non-empty asynchronous schedule 1 Empty asynchronous schedule
12	HCH	HC halted. This bit is a zero whenever USB_CMD[RS] is a one. The USB DR controller sets this bit to one after it has stopped executing because of USB_CMD[RS] being cleared, either by software or by the host controller hardware (for example, internal error). Only used in host mode. 0 Running 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) occurs.
9	—	Reserved, should be cleared.

Table 13-11. USBSTS Register Field Descriptions (continued)

Bits	Name	Description
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. 0 Active 1 Suspended
7	SRI	Host mode: <ul style="list-style-type: none"> <li>This is a non-EHCI status bit. In host mode, this bit is 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.</li> </ul> Device mode: <ul style="list-style-type: none"> <li>SOF received. When the USB DR controller detects a Start Of (Micro)Frame, this bit is set. When a SOF is extremely late, the DR controller automatically sets this bit to indicate that an SOF was expected. Therefore, this bit is set roughly every 1 msec in device FS mode and every 125 msec in HS mode and is synchronized to the actual SOF that is received. Because the controller is initialized to FS before connect, this bit is set at an interval of 1 msec during the prelude to the connect and chirp.</li> </ul> Software writes a 1 to this bit to clear it.
6	URI	USB reset received. This is a non-EHCI bit. When the USB DR controller detects a USB reset and enters the default state, this bit is set. Software can write a 1 to this bit to clear the USB reset received status bit. Only used by the device mode. 0 No reset received 1 Reset received
5	AAI	Interrupt on async advance. System software can force the controller to issue an interrupt the next time the USB DR 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. 0 No async advance interrupt 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 is generated. The interrupt and status bits remain asserted until cleared by writing a 1 to this bit. Additionally, when in host mode, USBCMD[RS] is cleared, effectively disabling the USB DR controller. For the USB DR controller in device mode, an interrupt is generated, but no other action is taken. 0 Normal operation 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 DR controller sets this bit to a one every time FHINDEX [12] toggles. Only used by the host mode.
2	PCI	Host mode: <ul style="list-style-type: none"> <li>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.</li> </ul> Device mode: <ul style="list-style-type: none"> <li>The USB DR 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.</li> </ul> This bit is not EHCI compatible.

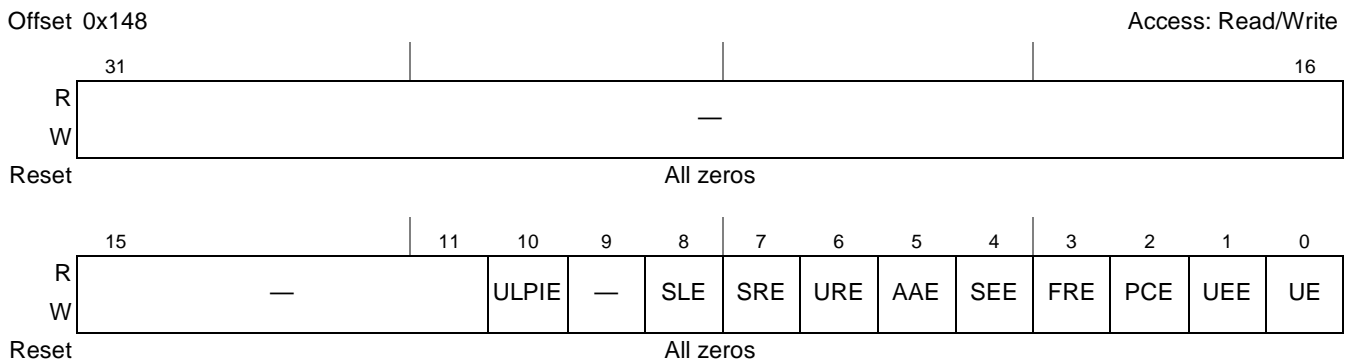


**Table 13-11. USBSTS Register Field Descriptions (continued)**

Bits	Name	Description
1	UEI (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 <a href="#">Table 13-91</a> in this chapter for more information on device error matrix. For the USB DR controller in device mode, only resume signaling is detected, all others are ignored. 0 No error 1 Error detected
0	UI (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.

### 13.3.2.3 USB Interrupt Enable Register (USBINTR)

The interrupts to software are enabled with the USB interrupt enable register, shown in [Figure 13-10](#). 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.


**Figure 13-10. USB Interrupt Enable (USBINTR)**

[Table 13-12](#) shows the USBINTR register field descriptions.

**Table 13-12. 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]) occurs. 0 Disable 1 Enable
9	—	Reserved, should be cleared.

Table 13-12. USBINTR Register Field Descriptions (continued)

Bits	Name	Description
8	SLE	Sleep enable. This is a non-EHCI bit. When this bit is a one, and USBSTS[SLI] transitions, the USB DR controller issues an interrupt. The interrupt is acknowledged by software writing a one to USBSTS[SLI]. Only used in device mode. 0 Disable 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 issues an interrupt. The interrupt is acknowledged by software clearing USBSTS[SRI]. 0 Disable 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 issues an interrupt. The interrupt is acknowledged by software clearing USBSTS[URI] bit. Only used in device mode. 0 Disable 1 Enable
5	AAE	Interrupt on async advance enable. When this bit is a one, and USBSTS[AAI] is a one, the controller issues an interrupt at the next interrupt threshold. The interrupt is acknowledged by software clearing USBSTS[AAI]. Only used in host mode. 0 Disable 1 Enable
4	SEE	System error enable. When this bit is a one, and USBSTS[SEI] is a one, the controller issues an interrupt. The interrupt is acknowledged by software clearing USBSTS[SEI]. 0 Disable 1 Enable
3	FRE	Frame list rollover enable. When this bit is a one, and USBSTS[FRI] is a one, the controller issues an interrupt. The interrupt is acknowledged by software clearing USBSTS[FRI]. Only used by the host mode. 0 Disable 1 Enable
2	PCE	Port change detect enable. When this bit is a one, and USBSTS[PCI] is a one, the controller issues an interrupt. The interrupt is acknowledged by software clearing USBSTS[PCI]. 0 Disable 1 Enable
1	UEE	USB error interrupt enable. When this bit is a one, and USBSTS[UEI] is a one, the controller issues an interrupt at the next interrupt threshold. The interrupt is acknowledged by software clearing USBSTS[UEI]. 0 Disable 1 Enable
0	UE	USB interrupt enable. When this bit is a one, and USBSTS[UI] is a one, the DR controller issues an interrupt at the next interrupt threshold. The interrupt is acknowledged by software clearing USBSTS[UI]. 0 Disable 1 Enable

### 13.3.2.4 Frame Index Register (FRINDEX)

In host mode, the frame index 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 DR 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 DR 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- $\mu$ sec microframe).

Figure 13-11 shows the USB frame index register.

Offset 0x14C

Access: Read/Write

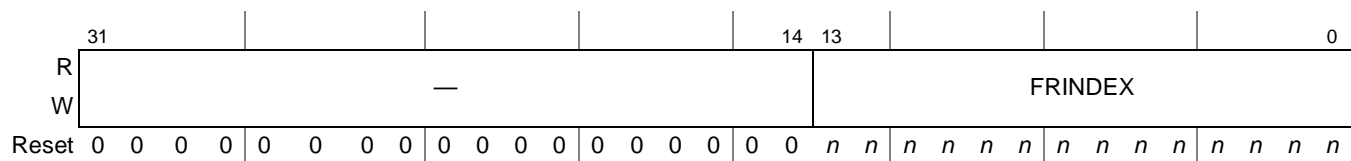


Figure 13-11. USB Frame Index (FRINDEX)

Table 13-13 shows the FRINDEX register field descriptions.

Table 13-13. 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). 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. In device mode, the value is the current frame number of the last frame transmitted. It is not used as an index. In either mode, bits 2–0 indicate the current microframe.

Table 13-14 illustrates values of N based on the value of the Frame List Size in the USBCMD register, when used in host mode.

Table 13-14. 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

**Table 13-14. FRINDEX N Values (continued)**

USBCMD[FS]	Frame List Size	FRINDEX N value
110	16 elements (64 bytes)	6
111	8 elements (32 bytes)	5

### 13.3.2.5 Control Data Structure Segment Register (CTRLDSSEGMENT)

The CTRLDSSEGMENT register is not implemented on MPC8308.

### 13.3.2.6 Periodic Frame List Base Address Register (PERIODICLISTBASE)

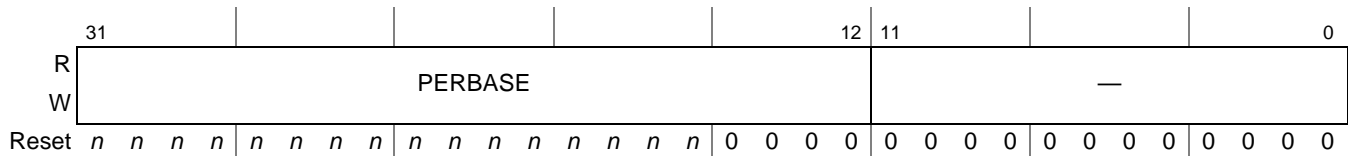
This register contains the beginning address of the Periodic Frame List in the system memory. The host controller driver loads this register prior to starting the schedule execution by the controller. The memory structure referenced by this physical memory pointer is assumed to be 4-Kbyte aligned. The 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 13.3.2.7, “Device Address Register \(DEVICEADDR\)—Non-EHCI,”](#) for more information.

[Figure 13-12](#) shows the periodic frame list base address register.

Offset 0x154

Access: Read/Write



**Figure 13-12. Periodic Frame List Base Address (PERIODICLISTBASE)**

[Table 13-15](#) shows the periodic frame list base address register field descriptions.

**Table 13-15. PERIODICLISTBASE Register Field Descriptions**

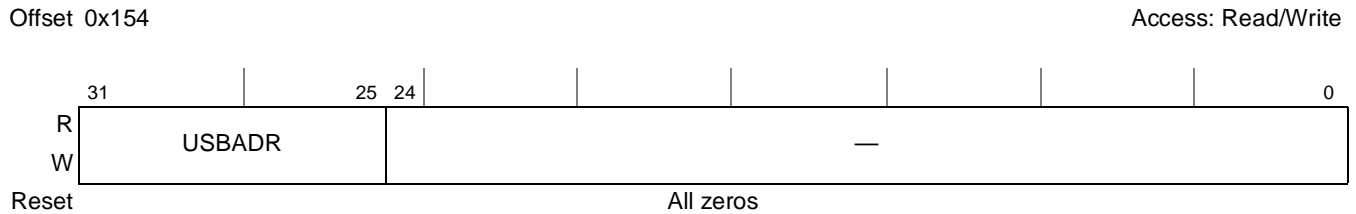
Bits	Name	Description
31–12	PERBASE	Base address. Correspond to memory address signal [31:12]. Only used in the host mode.
11–0	—	Reserved, should be cleared.

### 13.3.2.7 Device Address Register (DEVICEADDR)—Non-EHCI

The device address 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 13.3.2.6, “Periodic Frame List Base Address Register \(PERIODICLISTBASE\),”](#) for more information.

Figure 13-12 shows the device address register.



**Figure 13-13. Device Address (DEVICEADDR)**

Table 13-16 shows the device address register field descriptions.

**Table 13-16. DEVICEADDR Register Field Descriptions**

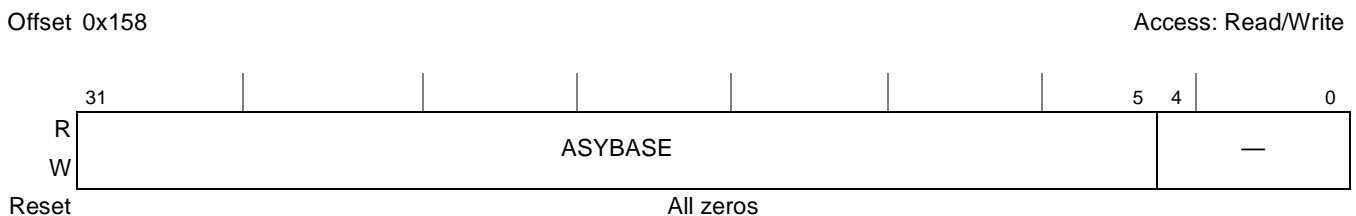
Bits	Name	Description
31–25	USBADR	Device address. This field corresponds to the USB device address.
24–0	—	Reserved, should be cleared.

### 13.3.2.8 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 13.3.2.9, “Endpoint List Address Register \(ENDPOINTLISTADDR\)—Non-EHCI,”](#) for more information.

Figure 13-14 shows the current asynchronous list address register.



**Figure 13-14. Current Asynchronous List Address (ASYNCLISTADDR)**

Table 13-17 describes the current asynchronous list address register.

**Table 13-17. ASYNCLISTADDR Register Field Descriptions**

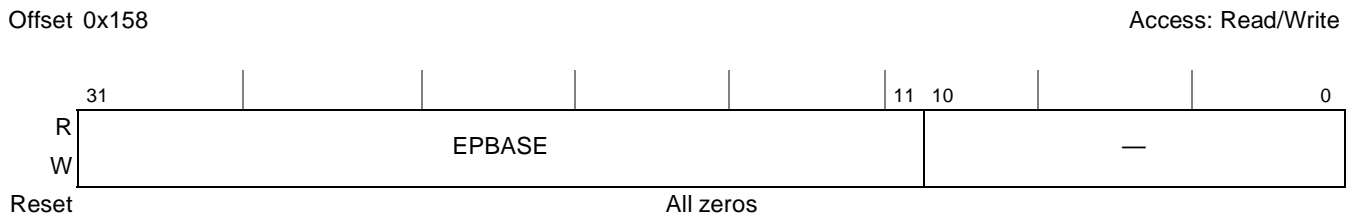
Bits	Name	Description
31–5	ASYBASE	Link pointer low (LPL). These bits correspond to memory address signals [31:5]. This field may only reference a queue head (QH). Only used by the host controller.
4–0	—	Reserved, should be cleared.

### 13.3.2.9 Endpoint List Address Register (ENDPOINTLISTADDR)—Non-EHCI

The endpoint list address 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 13.3.2.8, “Current Asynchronous List Address Register (ASYNCLISTADDR),” for more information.

Figure 13-15 shows the endpoint list address register.



**Figure 13-15. Endpoint List Address (ENDPOINTLISTADDR)**

Table 13-18 describes the endpoint list address register fields.

**Table 13-18. 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.

### 13.3.2.10 Master Interface Data Burst Size Register (BURSTSIZE)—Non-EHCI

The master interface data burst size register, shown in Figure 13-16, 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

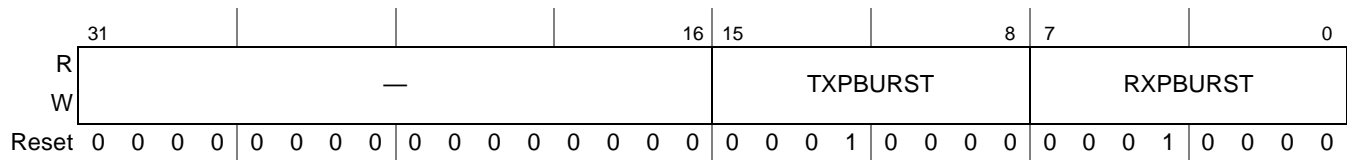

**Figure 13-16. Master Interface Data Burst Size (BURSTSIZE)**

Table 13-19 describes the master interface data burst size register fields.

**Table 13-19. BURSTSIZE Register Field Descriptions**

Bits	Name	Description
31–16	—	Reserved, should be cleared.
15–8	TXPBURST	Programmable 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 than 16.
7–0	RXPBURST	Programmable 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.

### 13.3.2.11 Transmit FIFO Tuning Controls Register (TXFILLTUNING)—Non-EHCI

The transmit FIFO tuning controls register, shown in Figure 13-17, 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 DR 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 TXSCHHEALTH ( $T_{ff}$ ) parameter described below.

Offset 0x164

Access: Read/Write

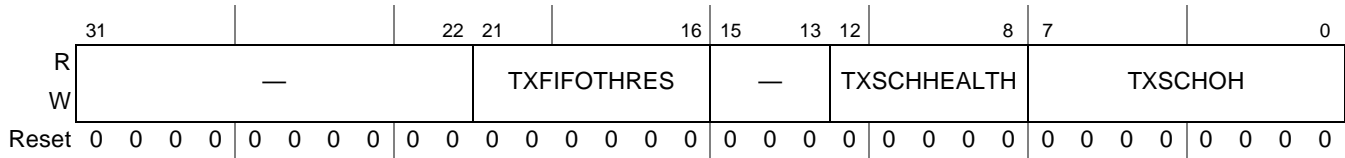


Figure 13-17. Transmit FIFO Tuning Controls (TXFILLTUNING)

Table 13-20 describes the transmit FIFO tuning controls register fields.

Table 13-20. TXFILLTUNING Register Field Descriptions

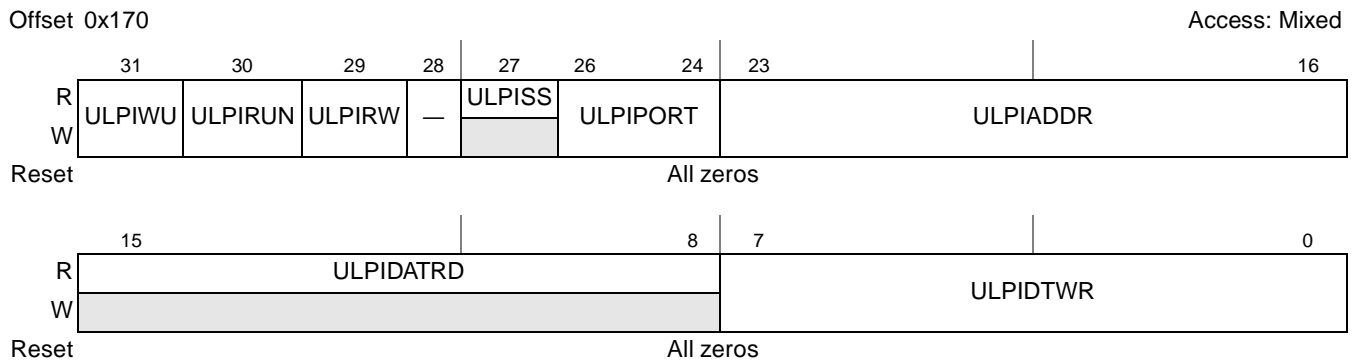
Bits	Name	Description
31–22	—	Reserved, should be cleared.
21–16	TXFIFOTHRES	FIFO burst threshold. Control the number of data bursts that are posted to the TX latency FIFO in host mode before the packet begins on to the bus. The minimum value is 2 and this value should be a low as possible to maximize 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. 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. The time unit represented in this register is 1.267 $\mu$ s when a device is connected in high-speed mode. The time unit represented in this register is 6.333 $\mu$ s when a device is connected in low-/full-speed mode. For most applications, TXSCHOH can be set to 4 or less. A good value to begin with is: $TXFIFOTHRES \times (BURSTSIZE \times 4 \text{ bytes-per-word}) \div (40 \times TimeUnit)$ , 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 $5 \times (8 \times 4) \div (40 \times 1.267) = 4$ 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. If streaming mode is disabled via the USBMODE register, treat TXFIFOTHRES as the maximum value for purposes of the TXSCHOH calculation.



### 13.3.2.12 ULPI Register Access (ULPI VIEWPORT)

The ULPI register access 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 through 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 13-18](#).



**Figure 13-18. ULPI Register Access (ULPI VIEWPORT)**

[Table 13-21](#) describes the ULPI register access fields.

**Table 13-21. 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. <b>Note:</b> 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. <b>Note:</b> 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. 0 Read 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. 0 Any other state (that is, carkit, serial, low power). 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.



Table 13-22 describes the configure flag register fields.

**Table 13-22. CONFIGFLAG Register Field Descriptions**

Bits	Name	Description
31–1	—	Reserved.
0	CF	Configure flag. Always 1 indicating all port routings default to this host.

### 13.3.2.14 Port Status and Control Register (PORTSC)

The port status and control (PORTSC) register, shown in Figure 13-20, 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 DR 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	0	0	0	1	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	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	0	0	0

**Figure 13-20. Port Status and Control (PORTSC)**

Table 13-23 describes the PORTSC register fields.

**Table 13-23. 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. 00 Reserved 01 Reserved 10 ULPI parallel interface 11 Reserved This bit is not defined in the EHCI specification.
29	—	Reserved, should be cleared
28	—	Reserved

Table 13-23. PORTSC Register Field Descriptions (continued)

Bits	Name	Description
27–26	PSPD	Port speed. This read-only register field indicates the speed at which the port is operating. This bit is not defined in the EHCI specification. 00 Full-speed 01 Low-speed 10 High-speed 11 Undefined
25	—	Reserved, should be cleared
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. 0 Allow the port to identify itself as high speed. 1 Force the port to only connect at full speed. This bit is not defined in the EHCI specification. This bit is for debugging purposes.
23	PHCD	PHY low power suspend. This bit is not defined in the EHCI specification. Host mode: <ul style="list-style-type: none"> <li>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.</li> </ul> Device mode: <ul style="list-style-type: none"> <li>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 is cleared automatically when the resume signaling has been detected or when forcing port resume.</li> </ul> 0 Normal PHY operation. 1 Signal the PHY to enter low power suspend mode Reading this bit indicates the status of the PHY. <b>Note:</b> If there is no clock connected to the USBDR_CLK signals, PHCD must be set and the following registers should not be written: DEVICE_ADDR/PERIODICLISTBASE, PORTSC, ENDPTCTRL0, ENDPTCTRL1, ENDPTCTRL2.
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. This field is zero if Port Power (PP) is zero. This bit is (OTG/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. This field is zero if Port Power(PP) is zero or in device mode. This bit is (OTG/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. This field is zero if Port Power(PP) is zero or in device mode. This bit is (OTG/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. 0000 Not Enabled 0001 J_STATE 0010 K_STATE 0011 SEQ_NAK 0100 Packet 0101 FORCE_ENABLE 0110–1111 Reserved, should be cleared Refer to Chapter 7 of the USB Specification Revision 2.0 [3] for details on each test mode.

Table 13-23. PORTSC Register Field Descriptions (continued)

Bits	Name	Description
15–14	PIC	Port indicator control. Control the link indicator signals. These signals are valid for host mode only. 00 Off 01 Amber 10 Green 11 Undefined Refer to the USB Specification Revision 2.0 [3] for a description on how these bits are to be used. This field is output from the module on the USB port control signals for use by an external LED driving circuit.
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. 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. 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). This feature is implemented in the host/OTG controller (PPC = 1). 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. 00 SE0 10 J-state 01 K-state 11 Undefined
9	—	Reserved, should be cleared
8	PR	Port reset. Host mode: <ul style="list-style-type: none"> <li>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.</li> </ul> Device mode: <ul style="list-style-type: none"> <li>This bit is a read only status bit. Device reset from the USB bus is also indicated in the USBSTS register.</li> </ul> 1 Port is in reset. 0 Port is not in reset. This field is zero if Port Power(PP) is zero.

Table 13-23. PORTSC Register Field Descriptions (continued)

Bits	Name	Description
7	SUSP	<p>Suspend.</p> <p>Host mode:</p> <ul style="list-style-type: none"> <li>The port enabled bit (PE) and suspend (SUSP) bit define the port states as follows:                      0x Disable                      10 Enable                      11 Suspend</li> <li>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.</li> <li>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.</li> <li>This field is zero if Port Power (PP) is zero in host mode.</li> </ul> <p>Device mode:</p> <p>1 Port in suspend state.                      0 Port not in suspend state. Default.</p> <p>In device mode this bit is a read-only status bit.</p>
6	FPR	<p>Force port resume. This bit is not-EHCI compatible.</p> <p>1 Resume detected/driven on port.                      0 No resume (K-state) detected/driven on port.</p> <p>Host mode:</p> <ul style="list-style-type: none"> <li>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.</li> <li>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.</li> <li>This field is zero if Port Power (PP) is zero in host mode.</li> </ul> <p>Device mode:</p> <ul style="list-style-type: none"> <li>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 DR 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 is 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.</li> </ul>
5	OCC	<p>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.</p> <p>Host/OTG mode:</p> <ul style="list-style-type: none"> <li>The user can provide over-current detection to the USB<sub>n</sub>_PWRFAULT signal for this condition.</li> </ul> <p>Device mode:</p> <ul style="list-style-type: none"> <li>This bit must always be 0.</li> </ul> <p>1 Over current detect.                      0 No over current.</p>

Table 13-23. PORTSC Register Field Descriptions (continued)

Bits	Name	Description
4	OCA	Over-current active. This bit will automatically transition from one to zero when the over current condition is removed. Host/OTG mode: <ul style="list-style-type: none"> <li>The user can provide over-current detection to the USB<sub>n</sub>_PWRFAULT signal for this condition.</li> </ul> Device mode: <ul style="list-style-type: none"> <li>This bit must always be 0.</li> </ul> 1 Port currently in over-current condition. 0 Port not in over-current condition.
3	PEC	Port enable/disable change. 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. In device mode: <ul style="list-style-type: none"> <li>The device port is always enabled. (This bit is zero.)</li> </ul> 1 Port disabled. 0 No change. This field is zero if Port Power(PP) is zero.
2	PE	Port enabled/disabled. Host mode: <ul style="list-style-type: none"> <li>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.</li> <li>When the port is disabled, (0) downstream propagation of data is blocked except for reset.</li> <li>This field is zero if Port Power(PP) is zero in host mode.</li> </ul> Device mode: <ul style="list-style-type: none"> <li>The device port is always enabled. (This bit is one.)</li> </ul>
1	CSC	Connect change status. Host mode: <ul style="list-style-type: none"> <li>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 is 'setting' an already-set bit (i.e., the bit will remain set). Software clears this bit by writing a one to it.</li> </ul> 1 Connect Status has changed. 0 No change. <ul style="list-style-type: none"> <li>This field is zero if Port Power(PP) is zero.</li> </ul> Device mode: <ul style="list-style-type: none"> <li>This bit is undefined.</li> </ul>
0	CCS	Current connect status. Host mode: 1 Device is present 0 No device present. This field is zero if Port Power(PP) is zero in host mode. In device mode: 1 Attached 0 Not attached. 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.

### 13.3.2.15 On-The-Go Status and Control (OTGSC)—Non-EHCI

This register is not defined in the EHCI specification. The USB DR module implements one On-The-Go (OTG) status and control register corresponding to Port 0.

The OTGSC register has four sections:

- OTG interrupt enables (Read/Write)
- OTG interrupt status (Read/Write to Clear)
- OTG status inputs (Read Only)
- OTG controls (Read/Write)

The status inputs are de-bounced using a 1-msec time constant. Values on the status inputs that do not persist for more than 1 msec will not cause an update of the status inputs, or cause an OTG interrupt.

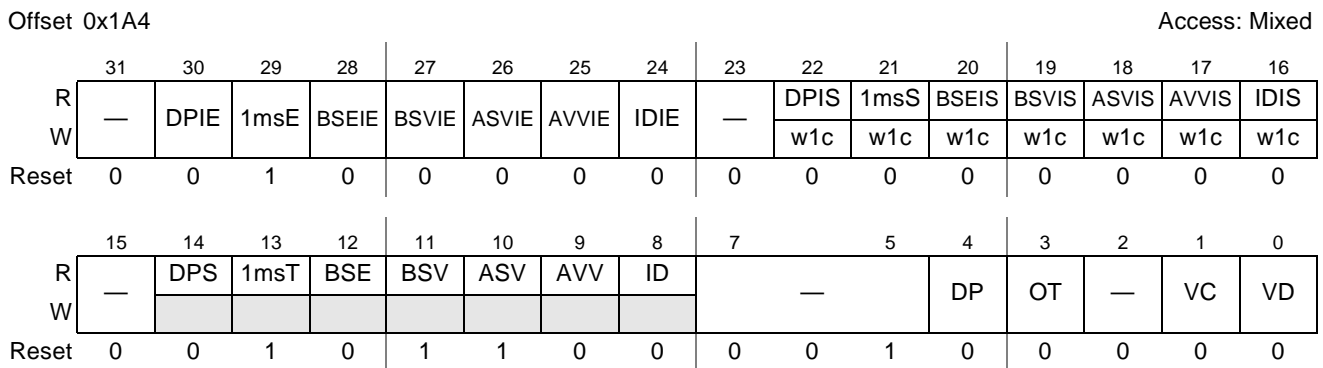


Figure 13-21. OTG Status Control (OTGSC)

Table 13-24. OTGSC Register Field Descriptions

Bits	Name	Description
31	—	Reserved, should be cleared.
30	DPIE	Data pulse interrupt enable 1 Enable 0 Disable
29	1msE	1-millisecond timer Interrupt enable 1 Enable 0 Disable
28	BSEIE	B session end interrupt enable 1 Enable 0 Disable
27	BSVIE	B session valid interrupt enable 1 Enable 0 Disable
26	ASVIE	A session valid interrupt enable 1 Enable 0 Disable



Table 13-24. OTGSC Register Field Descriptions (continued)

Bits	Name	Description
25	AVVIE	A VBus valid interrupt enable 1 Enable 0 Disable
24	IDIE	USB ID interrupt enable. 1 Enable 0 Disable
23	—	Reserved, should be cleared.
22	DPIS	Data pulse interrupt status. Set when data bus pulsing occurs on DP or DM. Data bus pulsing is only detected when USBMODE[CM] = Host (11) and PORTSC[PP] (port power) = Off (0). Software must write a one to clear this bit.
21	1msS	1-millisecond timer interrupt status. Set once every millisecond. Software must write a one to clear this bit.
20	BSEIS	B session end interrupt status. Set when VBus has fallen below the B session end threshold. Software must write a one to clear this bit.
19	BSVIS	B session valid interrupt status. Set when VBus has either risen above or fallen below the B session valid threshold (0.8 VDC). Software must write a one to clear this bit.
18	ASVIS	A session valid interrupt status. Set when VBus has either risen above or fallen below the A session valid threshold (0.8 VDC). Software must write a one to clear this bit.
17	AVVIS	A VBus valid interrupt status. Set when VBus has either risen above or fallen below the VBus valid threshold (4.4 VDC) on an A device. Software must write a one to clear this bit.
16	IDIS	USB ID interrupt status. Set when a change on the ID input has been detected. Software must write a one to clear this bit.
15	—	Reserved, should be cleared.
14	DPS	Data bus pulsing status 1 Pulsing detected on port 0 No pulsing on port
13	1msT	1 millisecond timer toggle. This bit toggles once per millisecond.
12	BSE	B session end 1 VBus is below the B session end threshold. 0 VBus is above the B session end threshold.
11	BSV	B session valid 1 VBus is above the B session valid threshold. 0 VBus is below the B session valid threshold.
10	ASV	A session valid 1 VBus is above the A session valid threshold. 0 VBus is below the A session valid threshold.
9	AVV	A VBus valid 1 VBus is above the A VBus valid threshold. 0 VBus is below the A VBus valid threshold.

Table 13-24. OTGSC Register Field Descriptions (continued)

Bits	Name	Description
8	ID	USB ID 1 B device 0 A device
7-5	—	Reserved, should be cleared.
4	DP	Data pulsing 1 The pullup on DP is asserted for data pulsing during SRP. 0 The pullup on DP is not asserted.
3	OT	OTG termination. This bit must be set when the OTG device is in device mode. 1 Enable pulldown on DM 0 Disable pulldown on DM
2	—	Reserved, should be cleared.
1	VC	VBUS charge. Setting this bit causes the VBus line to be charged. This is used for VBus pulsing during SRP.
0	VD	VBUS discharge. Setting this bit causes VBus to discharge through a resistor.

### 13.3.2.16 USB Mode Register (USBMODE)—Non-EHCI

The USB mode register, shown in Figure 13-22, is not defined in the EHCI specification. This register controls the operating mode of the module.



Figure 13-22. USB Mode (USBMODE)

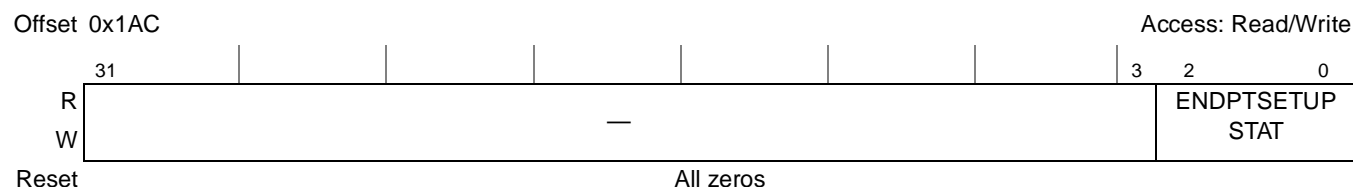
Table 13-25 describes the USB mode register fields.

**Table 13-25. USBMODE Register Field Descriptions**

Bits	Name	Description
31–5	—	Reserved, should be cleared.
4	SDIS	<p>Stream disable</p> <p>In host mode, 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.</p> <p>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.</p> <p>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.</p> <p>1 Active. 0 Inactive.</p> <p>In device mode, 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.</p> <p>Note that in high-speed mode, all packets received are responded to with a NYET handshake when stream disable is active.</p>
3	SLOM	<p>Setup lockout mode. In device mode, this bit controls behavior of the setup lock mechanism. See <a href="#">Section 13.8.3.5, “Control Endpoint Operation Model.”</a></p> <p>1 Setup lockouts off. DCD requires use of setup data buffer tripwire in USBCMD (SUTW). 0 Setup lockouts on</p>
2	—	Reserved, should be cleared.
1–0	CM	<p>Controller mode</p> <p>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.</p> <p>00 Idle (default for combination host/device). 01 Reserved, should be cleared. 10 Device controller (default for device only controller). 11 Host controller (default for host only controller).</p> <p>Defaults to the idle state and needs to be initialized to the desired operating mode after reset.</p>

### 13.3.2.17 Endpoint Setup Status Register (ENDPTSETUPSTAT)—Non-EHCI

The endpoint setup status register, shown in [Figure 13-23](#), is not defined in the EHCI specification. This register contains the endpoint setup status. It is only used in device mode.



**Figure 13-23. Endpoint Setup Status (ENDPTSETUPSTAT)**

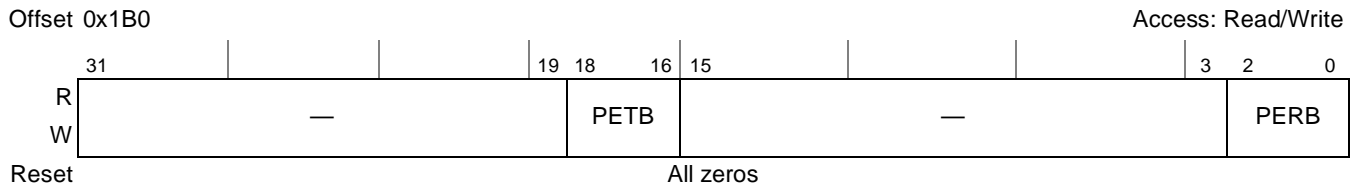
Table 13-26 describes the endpoint setup status register fields.

**Table 13-26. ENDPTSETUPSTAT Register Field Descriptions**

Bits	Name	Description
31–3	—	Reserved, should be cleared.
2–0	ENDPTSETUPSTAT	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. This register is only used in device mode.

### 13.3.2.18 Endpoint Initialization Register (ENDPTPRIME)—Non-EHCI

The endpoint initialization register, shown in Figure 13-23, is not defined in the EHCI specification. This register is used to initialize endpoints. It is only used in device mode.



**Figure 13-24. Endpoint Initialization (ENDPTPRIME)**

Table 13-27 describes the endpoint initialization register fields.

**Table 13-27. ENDPTPRIME Register Field Descriptions**

Bits	Name	Description
31–19	—	Reserved, should be cleared.
18–16	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[2] (bit 18 of the register) corresponds to endpoint 2. Note that these bits are momentarily set by hardware during hardware re-priming operations when a dTD is retired, and the dQH is updated.
15–3	—	Reserved, should be cleared.
2–0	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[2] corresponds to endpoint 2. Note that these bits are momentarily set by hardware during hardware re-priming operations when a dTD is retired, and the dQH is updated.

### 13.3.2.19 Endpoint Flush Register (ENDPTFLUSH)—Non-EHCI

The endpoint flush register, shown in Figure 13-25, is not defined in the EHCI specification. This register is only used in device mode.

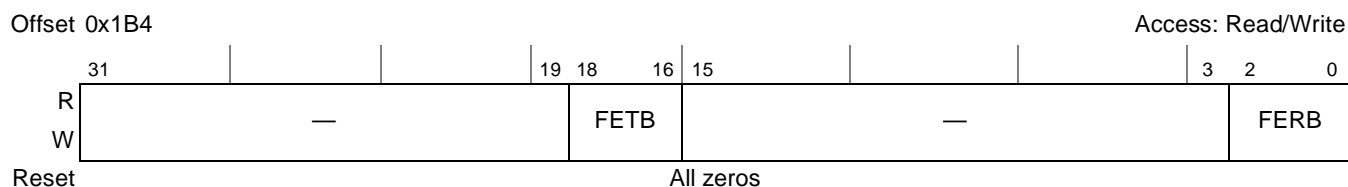


Figure 13-25. Endpoint Flush (ENDPTFLUSH)

Table 13-28 describes the endpoint flush register fields.

Table 13-28. ENDPTFLUSH Register Field Descriptions

Bits	Name	Description
31–19	—	Reserved, should be cleared.
18–16	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[2] (bit 18 of the register) corresponds to endpoint 2.
15–3	—	Reserved, should be cleared.
2–0	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[2] corresponds to endpoint 2.

### 13.3.2.20 Endpoint Status Register (ENDPTSTATUS)—Non-EHCI

The endpoint status register, shown in Figure 13-26, is not defined in the EHCI specification. This register is only used in device mode.



Figure 13-26. Endpoint Status (ENDPTSTATUS)

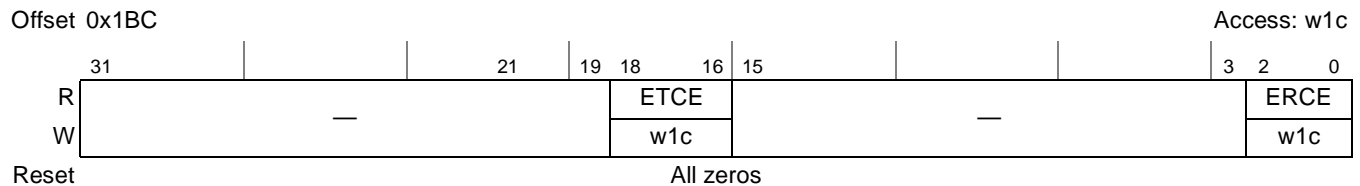
Table 13-29 describes the endpoint status fields.

**Table 13-29. ENDPTSTATUS Register Field Descriptions**

Bits	Name	Description
31–19	—	Reserved, should be cleared
18–16	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[2] (bit 18 of the register) corresponds to endpoint 2. Note that these bits are momentarily cleared by hardware during hardware endpoint re-priming operations when a dTD is retired, and the dQH is updated.
15–3	—	Reserved, should be cleared
2–0	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[2] corresponds to endpoint 2. Note that these bits are momentarily cleared by hardware during hardware endpoint re-priming operations when a dTD is retired, and the dQH is updated.

### 13.3.2.21 Endpoint Complete Register (ENDPTCOMPLETE)—Non-EHCI

The endpoint complete register, shown in Figure 13-25, is not defined in the EHCI specification. This register is only used in device mode.



**Figure 13-27. Endpoint Complete (ENDPTCOMPLETE)**

Table 13-30 describes the endpoint complete register fields.

**Table 13-30. ENDPTCOMPLETE Register Field Descriptions**

Bits	Name	Description
31–19	—	Reserved, should be cleared
18–16	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 is set simultaneously with the USBINT. Writing a one will clear the corresponding bit in this register. ETCE[2] (bit 18 of the register) corresponds to endpoint 2.

**Table 13-30. ENDPTCOMPLETE Register Field Descriptions (continued)**

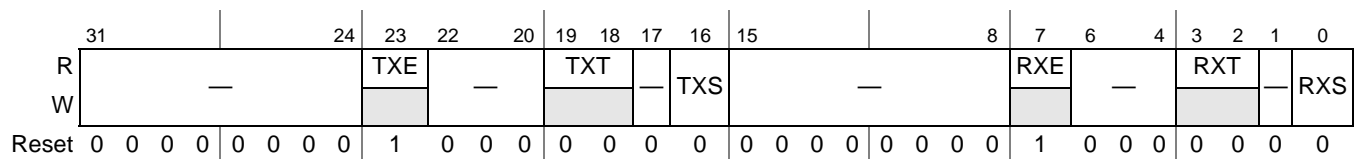
Bits	Name	Description
15–3	—	Reserved, should be cleared
2–0	ERCE	Endpoint receive complete event. Each bit indicates a received event (OUT/SETUP) occurred and software should read the corresponding endpoint queue to determine the transfer status. If the corresponding IOC bit is set in the Transfer Descriptor, then this bit is set simultaneously with the USBINT. Writing a one will clear the corresponding bit in this register. ERCE[2] corresponds to endpoint 2.

### 13.3.2.22 Endpoint Control Register 0 (ENDPTCTRL0)—Non-EHCI

Endpoint control register 0, shown in [Figure 13-25](#), is not defined in the EHCI specification. Every device will implement endpoint 0 as a control endpoint.

Offset 0x1C0

Access: Mixed


**Figure 13-28. Endpoint Control 0 (ENDPTCTRL0)**

[Table 13-31](#) describes the endpoint control register 0 fields.

**Table 13-31. ENDPTCTRL0 Register Field Descriptions**

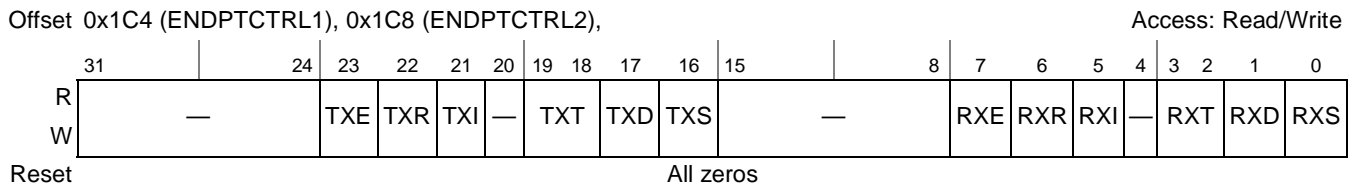
Bits	Name	Description
31–24	—	Reserved, should be cleared.
23	TXE	TX endpoint enable. Endpoint zero is always enabled. 0 Disable 1 Enable
22–20	—	Reserved, should be cleared.
19–18	TXT	TX endpoint type. Endpoint zero is always a control endpoint (00).
17	—	Reserved, should be cleared.
16	TXS	TX endpoint 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. 1 Endpoint stalled 0 Endpoint OK
15–8	—	Reserved, should be cleared.
7	RXE	RX endpoint enable. Endpoint zero is always enabled. 0 Disabled 1 Enabled
6–4	—	Reserved, should be cleared.
3–2	RXT	RX endpoint type. Endpoint zero is always a control endpoint (00).

**Table 13-31. ENDPTCTRL0 Register Field Descriptions (continued)**

Bits	Name	Description
1	—	Reserved, should be cleared.
0	RXS	RX 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. 1 Endpoint stalled 0 Endpoint OK

**13.3.2.23 Endpoint Control Register *n* (ENDPTCTRL*n*)—Non-EHCI**

The endpoint control *n* registers, shown in Figure 13-29, are not defined in the EHCI specification. There is an ENDPTCTRL*n* register of each endpoint in a device.



**Figure 13-29. Endpoint Control 1 to 5 (ENDPTCTRL*n*)**

Table 13-32 describes the endpoint control *n* register fields.

**Table 13-32. ENDPTCTRL*n* Register Field Descriptions**

Bits	Name	Description
31–24	—	Reserved, should be cleared
23	TXE	TX endpoint enable 0 Disabled 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. 0 PID sequencing enabled 1 PID sequencing disabled
20	—	Reserved, should be cleared
19–18	TXT	TX endpoint type 00 Control 01 Isochronous 10 Bulk 11 Interrupt <b>Note:</b> 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.



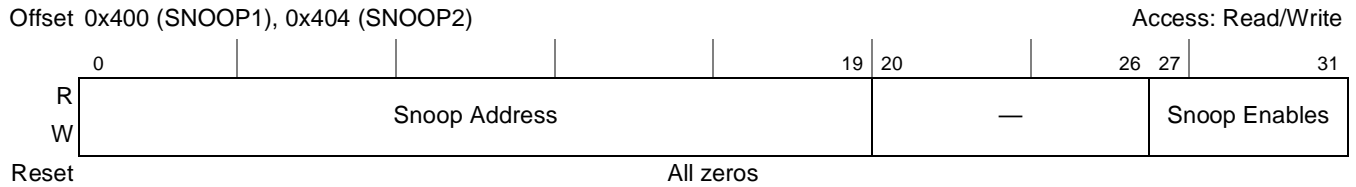
Table 13-32. ENDPTCTRL<sub>n</sub> Register Field Descriptions (continued)

Bits	Name	Description
16	TXS	TX endpoint stall. This bit is set automatically upon receipt of a SETUP request if this endpoint is not configured as a control endpoint. It is cleared automatically upon receipt of a SETUP request if this endpoint is configured as a control endpoint. 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. 0 Endpoint OK 1 Endpoint stalled
15–8	—	Reserved, should be cleared
7	RXE	RX endpoint enable 0 Disabled 1 Enabled
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. 1 PID sequencing enabled 0 PID sequencing disabled
4	—	Reserved, should be cleared
3–2	RXT	RX endpoint type 00 Control 01 Isochronous 10 Bulk 11 Interrupt <b>Note:</b> 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 is set automatically upon receipt of a SETUP request if this endpoint is not configured as a control endpoint. It is cleared automatically upon receipt a SETUP request if this endpoint is configured as a control endpoint, 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, 1 Endpoint stalled 0 Endpoint OK

### 13.3.2.24 SNOOP1 and SNOOP2—Non-EHCI

Figure 13-30 shows the SNOOP1 and SNOOP2 registers. 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 CSB bus. When the five lower bits (SNOOP<sub>n</sub>[27–31]) are equal to 00000, snooping is always disabled on the CSB for all DMA transfers. When SNOOP<sub>n</sub>[27–31] is 01011 through 11110, the twenty upper bits (SNOOP<sub>n</sub>[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<sub>n</sub>[20–26] are not used.



**Figure 13-30. Snoop 1 and Snoop 2 (SNOOP<sub>n</sub>)**

Table 13-33 describes the SNOOP<sub>n</sub> register fields.

**Table 13-33. SNOOP<sub>n</sub> 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 0x0B 4-Kbyte snoop range starting at the value defined by SNOOP <sub>n</sub> [0–19] 0x0C 8-Kbyte snoop range starting at the value defined by SNOOP <sub>n</sub> [0–18] 0x0D 16-Kbyte snoop range starting at the value defined by SNOOP <sub>n</sub> [0–17] 0x0E 32-Kbyte snoop range starting at the value defined by SNOOP <sub>n</sub> [0–16] 0x0F 64-Kbyte snoop range starting at the value defined by SNOOP <sub>n</sub> [0–15] 0x10 128-Kbyte snoop range starting at the value defined by SNOOP <sub>n</sub> [0–14] 0x11 256-Kbyte snoop range starting at the value defined by SNOOP <sub>n</sub> [0–13] 0x12 512-Kbyte snoop range starting at the value defined by SNOOP <sub>n</sub> [0–12] 0x13 1-Mbyte snoop range starting at the value defined by SNOOP <sub>n</sub> [0–11] 0x14 2-Mbyte snoop range starting at the value defined by SNOOP <sub>n</sub> [0–10] 0x15 4-Mbyte snoop range starting at the value defined by SNOOP <sub>n</sub> [0–9] 0x16 8-Mbyte snoop range starting at the value defined by SNOOP <sub>n</sub> [0–8] 0x17 16-Mbyte snoop range starting at the value defined by SNOOP <sub>n</sub> [0–7] 0x18 32-Mbyte snoop range starting at the value defined by SNOOP <sub>n</sub> [0–6] 0x19 64-M byte snoop range starting at the value defined by SNOOP <sub>n</sub> [0–5] 0x1A 31-Mbyte snoop range starting at the value defined by SNOOP <sub>n</sub> [0–4] 0x1B 256-Mbyte snoop range starting at the value defined by SNOOP <sub>n</sub> [0–3] 0x1C 512-Mbyte snoop range starting at the value defined by SNOOP <sub>n</sub> [0–2] 0x1D 1-Gbyte snoop range starting at the value defined by SNOOP <sub>n</sub> [0–1] 0x1E 2-Gbyte snoop range starting at the value defined by SNOOP <sub>n</sub> [0]

### 13.3.2.25 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 DR controller’s internal system interface. This is used to increase the priority state of the module’s system interface from zero to one. The actual priority level on the system bus for each state is defined by the PRI\_CTRL register. See Section 6.3.1.1, “Address Bus Arbitration with PRIORITY[0:1],” for more details on bus priority. The threshold value is in units of *csb\_clk* 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, priority state zero is chosen. If the aging counter is greater than or equal to the AGE\_CNT\_THRESH value, priority state one 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, priority state one is always chosen. If the aging counter is less than the AGE\_CNT\_THRESH value, priority state zero is selected. If the aging counter is greater than or equal to the AGE\_CNT\_THRESH value, priority state one is selected.

The two priority states of the aging counter function each have corresponding register bits which are programmed by the CPU. Thus, when the aging counter function is at priority state zero, PRI\_CTRL[30–31] are selected and used to drive bus priority levels. When the aging counter function is at priority state one, PRI\_CTRL[28–29] are selected and used to drive the priority.

Figure 13-31 shows the age count threshold register.

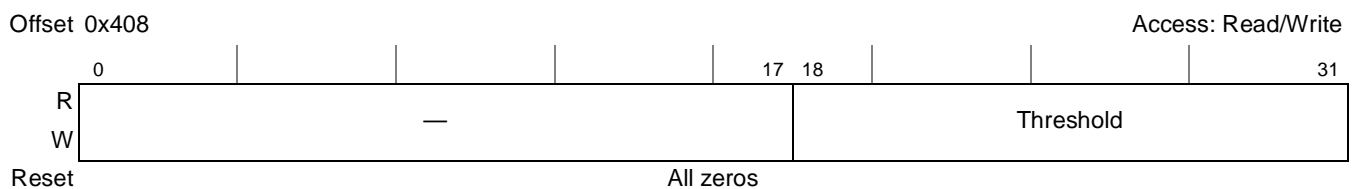


Figure 13-31. Age Count Threshold (AGE\_CNT\_THRESH)

Table 13-34 describes the age count threshold register fields.

Table 13-34. 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 settings:

- Set PRI\_CTRL[pri\_lv10] to 0.
- Set tPRI\_CTRL[pri\_lv11] to 3.
- Set AGE\_CNT\_THRESH to 40.

This combination works for a wide variety of applications. If this combination still does not meet application requirements, try lowering AGE\_CNT\_THRESH by 5. On the contrary, the setting 40 may be too conservative for some applications. If USB performance is acceptable at 40, try raising the value in

increments of 5. 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.

### 13.3.2.26 Priority Control Register (PRI\_CTRL)—Non-EHCI

Note that this register uses big-endian byte ordering and is not defined in the EHCI specification. The priority control (PRI\_CTRL) register sets the priority level for each of two priority states. The priority state is determined by the value programmed in the AGE\_CNT\_THRESH register and the number of *csb\_clk* cycles that a particular transaction takes to complete.

Figure 13-32 shows the priority control register.

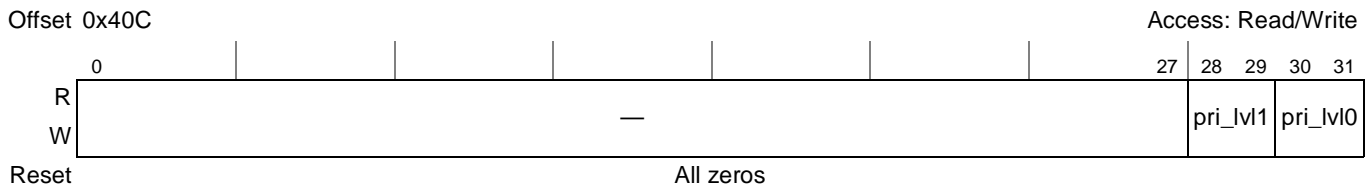


Figure 13-32. Priority Control (PRI\_CTRL)

Table 13-35 describes the priority control register fields.

Table 13-35. PRI\_CTRL Register Field Descriptions

Bits	Name	Description
0–27	—	Reserved, should be cleared
28–29	pri_lv1	Priority level for priority state 1.
30–31	pri_lv0	Priority level for priority state 0.

### 13.3.2.27 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.

Figure 13-33 shows the system interface control register.

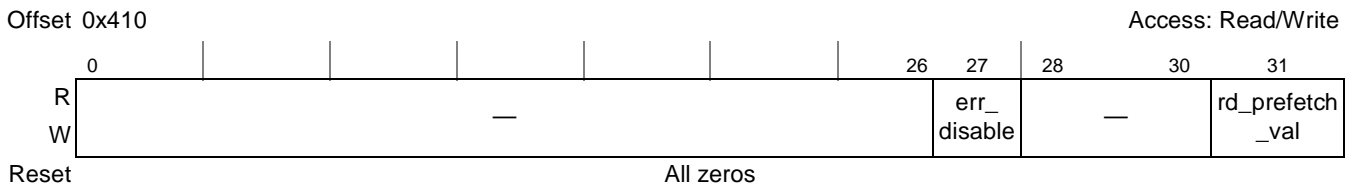


Figure 13-33. System Interface Control Register (SI\_CTRL)

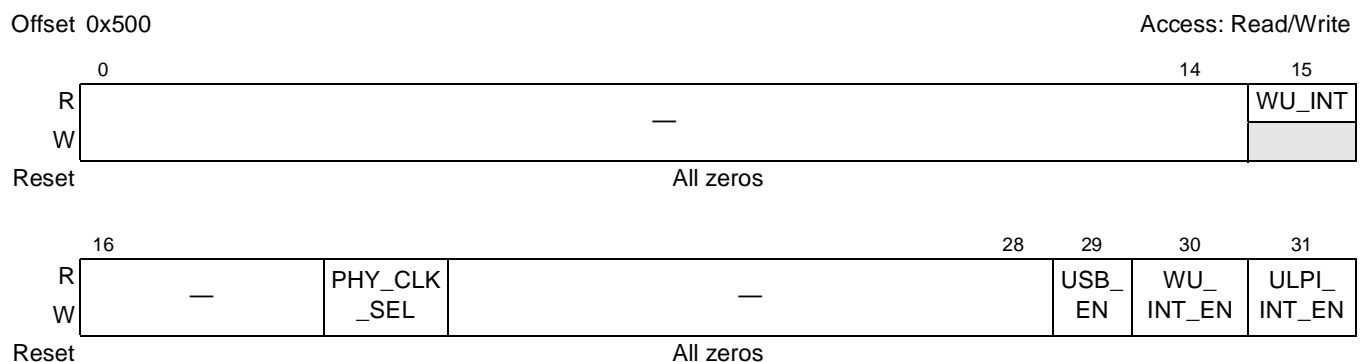
Table 13-36 describes the system interface control register fields.

**Table 13-36. 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]. 0 enable 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. 0 64-byte fetch 1 32-byte fetch

### 13.3.2.28 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 13-34.



**Figure 13-34. USB General-Purpose Register (CONTROL)**

Table 13-37 describes the USB general-purpose register fields.

**Table 13-37. 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. 0 Normal operation or Low Power mode waiting for wakeup event 1 Low power wakeup event has occurred
16–20	—	Reserved
21	PHY_CLK_SEL	Select the source of the USB link controller transceiver clock. When cleared the UTMI PHY is the source of the clock. When set, the clock is sourced from the external ULPI PHY. 0 UTMI is clock source 1 ULPI is clock source
22–28	—	Reserved
29	USB_EN	UTMI mode: This bit is used to enable the USB interface. It must be set before setting RS bit in USB CMD register. 1 Enable 0 Disable  ULPI mode: 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 DIR is forced to appear high 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. 1 Normal operation 0 Safe mode
30	WU_INT_EN	This bit is used to mask/unmask the system wakeup interrupt signal 0 System wakeup interrupt disabled 1 System wakeup interrupt enabled <b>Note:</b> 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. 0 ULPI low power wakeup interrupt disabled 1 ULPI low power wakeup interrupt enabled <b>Note:</b> PORTSC[PHCD] bit must be set

## 13.4 Functional Description

The USB DR module can be broken down into functional sub-blocks, which are described below.

### 13.4.1 System Interface

The system interface block contains all the control and status registers that allow a processor to interface to the USB DR 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.

### 13.4.2 DMA Engine

The module contains a local DMA engine. The DMA engine interfaces internally to the CSB. 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.

### 13.4.3 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.

### 13.4.4 PHY Interface

The USB DR 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 pin count limitations the module only supports certain combinations of PHY interfaces and USB functionality. Refer to [Table 13-38](#) for more information.

**Table 13-38. Supported PHY Interfaces**

PHY	Function
ULPI	Host/Device/OTG

## 13.5 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.

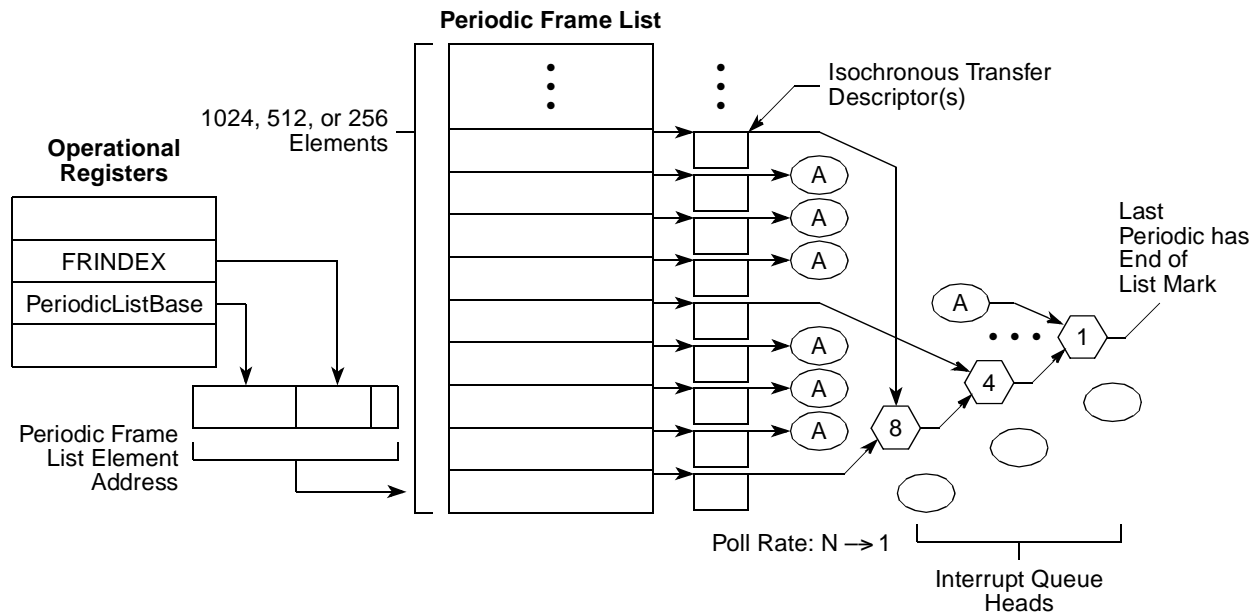
Note that software must ensure that no interface data structure reachable by the EHCI host controller spans a 4K-page boundary.

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.

### 13.5.1 Periodic Frame List

Figure 13-35 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.





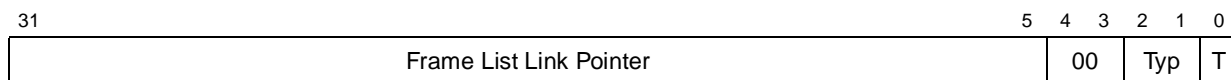
**Figure 13-35. Periodic Schedule Organization**

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 microframe. The link pointers are aligned on DWord boundaries within the frame list.

Figure 13-36 shows the format for the frame list link pointer.



**Figure 13-36. 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.

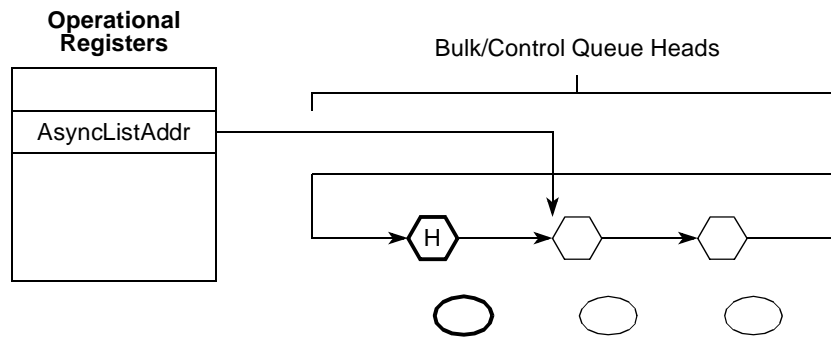
The least-significant bit is the T bit (bit 0). When this bit is set, the host controller never uses the value of the frame list pointer as a physical memory pointer. The Typ field indicates the exact type of data structure being referenced by this pointer. The value encodings for the Typ field are given in Table 13-39.

**Table 13-39. Typ Field Encodings**

Typ	Description
00	Isochronous transfer descriptor
01	Queue head
10	Split transaction isochronous transfer descriptor
11	Frame span traversal node

### 13.5.2 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 13-37 shows the asynchronous schedule organization.



**Figure 13-37. 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.

### 13.5.3 Isochronous (High-Speed) Transfer Descriptor (iTd)

Figure 13-38 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.

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
Next Link Pointer																												00	Typ	T	0x00	
Status <sup>1</sup>		Transaction 0 Length <sup>1</sup>										ioc	PG <sup>2</sup>	Transaction 0 Offset <sup>2</sup>										0x04								
Status <sup>1</sup>		Transaction 1 Length <sup>1</sup>										ioc	PG <sup>2</sup>	Transaction 1 Offset <sup>2</sup>										0x08								
Status <sup>1</sup>		Transaction 2 Length <sup>1</sup>										ioc	PG <sup>2</sup>	Transaction 2 Offset <sup>2</sup>										0x0C								
Status <sup>1</sup>		Transaction 3 Length <sup>1</sup>										ioc	PG <sup>2</sup>	Transaction 3 Offset <sup>2</sup>										0x10								
Status <sup>1</sup>		Transaction 4 Length <sup>1</sup>										ioc	PG <sup>2</sup>	Transaction 4 Offset <sup>2</sup>										0x14								
Status <sup>1</sup>		Transaction 5 Length <sup>1</sup>										ioc	PG <sup>2</sup>	Transaction 5 Offset <sup>2</sup>										0x18								
Status <sup>1</sup>		Transaction 6 Length <sup>1</sup>										ioc	PG <sup>2</sup>	Transaction 6 Offset <sup>2</sup>										0x1C								
Status <sup>1</sup>		Transaction 7 Length <sup>1</sup>										ioc	PG <sup>2</sup>	Transaction 7 Offset <sup>2</sup>										0x20								
Buffer Pointer (Page 0)											EndPt	R	Device Address										0x24									
Buffer Pointer (Page 1)											I/O	Maximum Packet Size										0x28										
Buffer Pointer (Page 2)											Reserved										Mult	0x2C										
Buffer Pointer (Page 3)											Reserved										0x30											
Buffer Pointer (Page 4)											Reserved										0x34											
Buffer Pointer (Page 5)											Reserved										0x38											
Buffer Pointer (Page 6)											Reserved										0x3C											

**Figure 13-38. Isochronous Transaction Descriptor (iTD)**

<sup>1</sup> Host controller read/write; all others read-only.

<sup>2</sup> These fields may be modified by the host controller if the I/O field indicates an OUT.

### 13.5.3.1 Next Link Pointer

The first DWord of an iTD is a pointer to the next schedule data structure, as shown in [Table 13-40](#).

**Table 13-40. 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: 00 iTD (isochronous transfer descriptor) 01 QH (queue head) 10 siTD (split transaction isochronous transfer descriptor) 11 FSTN (frame span traversal node)
0	T	Terminate 1 Link Pointer field is not valid. 0 Link Pointer field is valid.

### 13.5.3.2 iTD Transaction Status and Control List

DWords 1–8 constitute eight slots of transaction control and status. Each transaction description includes the following:

- 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 13-41 shows the iTD transaction status and control fields.

**Table 13-41. 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: 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. 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. 29 Babble detected. Set by the host controller during status update when "babble" is detected during the transaction generated by this descriptor. 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 <i>n</i> 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 <i>n</i> 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.

### 13.5.3.3 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)  $\times$  1024 (maximum packet size)  $\times$  8 (transaction records) = 24 576 bytes to be moved with this data structure, regardless of the alignment offset of the first page.

Since each pointer is a 4 K-aligned page pointer, the least-significant 12 bits in several of the page pointers are used for other purposes.

Table 13-42–Table 13-45 describes buffer pointer page *n*.

**Table 13-42. 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.
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 13-43. 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. 0 OUT 1 IN
10–0	Maximum Packet Size	This directly corresponds to the maximum packet size of the associated endpoint ( <i>wMaxPacketSize</i> ). This field is used for high-bandwidth endpoints where more than one transaction is issued per transaction description (for example, per microframe). This field is used with the <i>Multi</i> 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 13-44. 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 microframe). 00 Reserved, should be cleared. A zero in this field yields undefined results. 01 One transaction to be issued for this endpoint per microframe 10 Two transactions to be issued for this endpoint per microframe 11 Three transactions to be issued for this endpoint per microframe

**Table 13-45. 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.

### 13.5.4 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.

Figure 13-39 shows the split-transaction isochronous transfer descriptor (siTD).

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			
Next Link Pointer										00	Typ	T	0x00	
I/O	Port Number			0	Hub Address			0000	EndPt	0	Device Address			0x04
0000_0000_0000_00000						µFrame C-mask			µFrame S-mask			0x08		
ioc	P <sup>1</sup>	0000		Total Bytes to Transfer <sup>1</sup>			µFrame C-prog-mask <sup>1</sup>			Status <sup>1</sup>			0x0C	
Buffer Pointer (Page 0)						Current Offset <sup>1</sup>						0x10		
Buffer Pointer (Page 1)						000_0000			TP <sup>1</sup>	T-count <sup>1</sup>			0x14	
Back Pointer										0000		T	0x18	

**Figure 13-39. Split-Transaction Isochronous Transaction Descriptor (siTD)**

<sup>1</sup> Host controller read/write; all others read-only.

#### 13.5.4.1 Next Link Pointer

DWord0 of a siTD is a pointer to the next schedule data structure. Table 13-46 describes the next link pointer fields.

**Table 13-46. 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: 00 iTD (isochronous transfer descriptor) 01 QH (queue head) 10 siTD (split transaction isochronous transfer descriptor) 11 FSTN (frame span traversal node)
0	T	Terminate. 0 Link pointer is valid. 1 Link pointer field is not valid.

### 13.5.4.2 siTD Endpoint Capabilities/Characteristics

DWords 1 and 2 specify static information about the full-speed endpoint, the addressing of the parent Companion Controller, and microframe scheduling control.

Table 13-47 describes the endpoint and transaction translator characteristics.

**Table 13-47. 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. 0 OUT 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.
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 13-41 describes the microframe schedule control.

**Table 13-48. Microframe 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 microframes 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 microframes 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.

### 13.5.4.3 siTD Transfer State

DWords 3–6 manage the state of the transfer, as described in [Table 13-49](#).

**Table 13-49. siTD Transfer Status and Control**

Bits	Name	Description	
31	ioc	Interrupt on complete 0 Do not interrupt when transaction is complete. 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 0 Selects Page 0 pointer 1 Selects Page 1 pointer 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.	
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:	
		<b>Status Bits</b>	<b>Definition</b>
		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 microframe. 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: 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. 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.



### 13.5.4.4 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 13-50 describes the siTD buffer pointer page 0.

**Table 13-50. 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 13-51 describes the siTD buffer pointer page 1.

**Table 13-51. 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: 00 All. The entire full-speed transaction data payload is in this transaction (that is, less than or equal to 188 bytes). 01 Begin. This is the first data payload for a full-speed transaction that is greater than 188 bytes. 10 Mid. This is the middle payload for a full-speed OUT transaction that is larger than 188 bytes. 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.

### 13.5.4.5 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 13-52 describes the siTD back link pointer.

**Table 13-52. siTD Back Link Pointer**

Bits	Name	Description
31–5	Back Pointer	A physical memory pointer to an siTD

**Table 13-52. siTD Back Link Pointer (continued)**

Bits	Name	Description
4–1	—	Reserved, should be cleared. This field is reserved for future use. It should be cleared.
0	T	Terminate 0 siTD Back Pointer field is valid 1 siTD Back Pointer field is not valid

### 13.5.5 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 20,480 (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.

Figure 13-40 shows the queue element transfer descriptors.

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
Next qTD Pointer																										0000	T	0x00				
Alternate Next qTD Pointer																										0000	T	0x04				
dt <sup>1</sup>	Total Bytes to Transfer <sup>1</sup>														ioc	C_Page <sup>1</sup>	Cerr <sup>1</sup>	PID Code	Status <sup>1</sup>						0x08							
Buffer Pointer (Page 0)											Current Offset <sup>1</sup>						0x0C															
Buffer Pointer (Page 1)											0000_0000_0000						0x10															
Buffer Pointer (Page 2)											0000_0000_0000						0x14															
Buffer Pointer (Page 3)											0000_0000_0000						0x18															
Buffer Pointer (Page 4)											0000_0000_0000						0x1C															

**Figure 13-40. Queue Element Transfer Descriptor (qTD)**

<sup>1</sup> Host controller read/write; all others read-only.

Queue element transfer descriptors must be aligned on 32-byte boundaries.

### 13.5.5.1 Next qTD Pointer

The first DWord of an element transfer descriptor is a pointer to another transfer element descriptor. [Table 13-53](#) describes the qTD next element transfer pointer.

**Table 13-53. 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. 0 Pointer is valid (points to a valid transfer element descriptor) 1 Pointer is invalid

### 13.5.5.2 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 13-54](#) describes the alternate qTD next element transfer pointer.

**Table 13-54. 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. 0 Pointer is valid (points to a valid transfer element descriptor) 1 Pointer is invalid

### 13.5.5.3 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 13-55](#) reference fields are defined in the queue head. See [Section 13.5.6, “Queue Head,”](#) for more information on these fields.

**Table 13-55. 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 13-55. qTD Token (DWord 2) (continued)

Bits	Name	Description	
11–10	Cerr	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 setup, 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 is generated if USBINTR[UEE] is set. If the host controller driver (HCD) software programs this field to zero during setup, the host controller will not count errors for this qTD and there is 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.	
		<b>Error</b>	<b>Decrement Counter</b>
		Transaction Error	Yes
		Data Buffer Error	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.
		Stalled	No. Detection of babble or stall automatically halts the queue head. Thus, count is not decremented
		Babble Detected	No. Detection of babble or stall automatically halts the queue head. Thus, count is not decremented
		No Error	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.
9–8	PID Code	This field is an encoding of the token, which should be used for transactions associated with this transfer descriptor. Encodings are: 00 OUT Token generates token (E1H) 01 IN Token generates token (69H) 10 SETUP Token generates token (2DH) (undefined if endpoint is an Interrupt transfer type, for example, $\mu$ Frame S-mask field in the queue head is non-zero.) 11 Reserved, should be cleared	

**Table 13-55. qTD Token (DWord 2) (continued)**

Bits	Name	Description	
7-0	Status	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:	
		<b>Bits</b>	<b>Status Field Description</b>
		7	Active. Set by software to enable the execution of transactions by the host controller.
		6	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.
		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). 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.
		4	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.
		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). If the host controller sets this bit to a one, then it remains a one for the duration of the transfer.
		2	Missed microframe. 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.

Table 13-55. qTD Token (DWord 2) (continued)

Bits	Name	Description
1		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: 0 Do start split. This value directs the host controller to issue a start split transaction to the endpoint. 1 Do complete split. This value directs the host controller to issue a Complete split transaction to the endpoint.
0		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: 0 Do OUT. This value directs the host controller to issue an OUT PID to the endpoint. 1 Do Ping. This value directs the host controller to issue a PING PID to the endpoint. 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.

#### 13.5.5.4 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 13-56 describes the qTD buffer pointer.

Table 13-56. qTD Buffer Pointer

Bits	Name	Description
31–12	Buffer Pointer (page <i>n</i> )	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)/ — (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.

### 13.5.6 Queue Head

Figure 13-41 shows the queue head structure.

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
Queue Head Horizontal Link Pointer																											00	Typ	T	0x00		
RL			C	Maximum Packet Length				H	drc	EPS	EndPt		I	Device Address								0x04 <sup>1</sup>										
Mult		Port Number			Hub Addr			µFrame C-mask				µFrame S-mask								0x08 <sup>1</sup>												
Current qTD Pointer <sup>2</sup>														00000								0x0C										
Next qTD Pointer <sup>2</sup>														0000				T <sup>2</sup>				0x10 <sup>3</sup>										
Alternate Next qTD Pointer <sup>2</sup>														NakCnt <sup>2</sup>				T <sup>2</sup>				0x14 <sup>3,4</sup>										
dt <sup>1</sup>	Total Bytes to Transfer <sup>2</sup>						ioc <sup>2</sup>	C_Page <sup>2</sup>	Cerr <sup>2</sup>	PID Code <sup>2</sup>	Status <sup>2</sup>						0x18 <sup>3,4</sup>															
Buffer Pointer (Page 0) <sup>2</sup>									Current Offset <sup>2</sup>									0x1C <sup>3,4</sup>														
Buffer Pointer (Page 1) <sup>2</sup>									0000			C-prog-mask <sup>2</sup>						0x20 <sup>3,4</sup>														
Buffer Pointer (Page 2) <sup>2</sup>									S-bytes <sup>2</sup>						FrameTag <sup>2</sup>			0x24 <sup>3,4</sup>														
Buffer Pointer (Page 3) <sup>2</sup>									0000_0000_0000									0x28 <sup>3</sup>														
Buffer Pointer (Page 4) <sup>2</sup>									0000_0000_0000									0x2C <sup>3</sup>														

Figure 13-41. Queue Head Layout

- <sup>1</sup> Offsets 0x04 through 0x0B contain the static endpoint state.
- <sup>2</sup> Host controller read/write; all others read-only.
- <sup>3</sup> Offsets 0x10 through 0x2F contain the transfer overlay.
- <sup>4</sup> Offsets 0x14 through 0x27 contain the transfer results.

#### 13.5.6.1 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 13-57 describes the queue head.

Table 13-57. 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.



Table 13-57. Queue Head DWord 0 (continued)

Bits	Name	Description
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. 00 iTD (isochronous transfer descriptor) 01 QH (queue head) 10 siTD (split transaction isochronous transfer descriptor) 11 FSTN (frame span traversal node)
0	T	Terminate. 1 Last QH (pointer is invalid). 0 Pointer is valid. 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.

### 13.5.6.2 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 13-58 and Table 13-59 describe the endpoint characteristics.

Table 13-58. 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.

**Table 13-58. Endpoint Characteristics: Queue Head DWord 1 (continued)**

Bits	Name	Description
14	dtc	Data toggle control (DTC). Specifies where the host controller should get the initial data toggle on an overlay transition. 0 Ignore DT bit from incoming qTD. Host controller preserves DT bit in the queue head. 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. 00 Full-speed (12 Mbps) 01 Low-speed (1.5 Mbps) 10 High-speed (480 Mbps) 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 13-59. 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). 00 Reserved, should be cleared. A zero in this field yields undefined results. 01 One transaction to be issued for this endpoint per microframe 10 Two transactions to be issued for this endpoint per microframe 11 Three transactions to be issued for this endpoint per microframe
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 13-59. 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 microframes 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.

### 13.5.6.3 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 13-59 describes the current qTD link pointer.

**Table 13-60. 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.

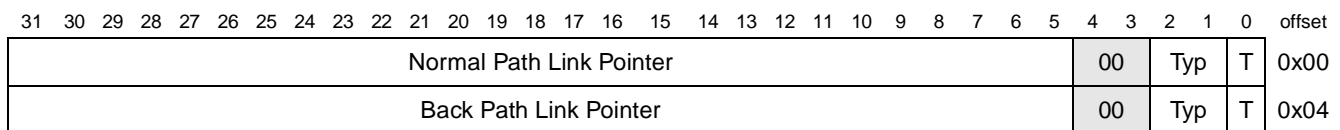
Table 13-61 describes the host-controller rules for bits in overlay.

**Table 13-61. 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.

### 13.5.7 Periodic Frame Span Traversal Node (FSTN)

The periodic frame span traversal node (FSTN) data structure, shown in Figure 13-42, 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.



**Figure 13-42. Frame Span Traversal Node Structure**

### 13.5.7.1 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 13-62](#) describes the FTSN normal path pointer.

**Table 13-62. 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. 00 iTD (isochronous transfer descriptor) 01 QH (queue head) 10 siTD (split transaction isochronous transfer descriptor) 11 FSTN (frame span traversal node)
0	T	Terminate. 0 Link pointer is valid. 1 Link pointer field is not valid.

### 13.5.7.2 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 13-63](#) describes the FTSN back path link pointer.

**Table 13-63. 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. 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. 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.

## 13.6 Host Operations

The general operational model for the USB DR module in host mode is defined by the 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.

### 13.6.1 Host Controller Initialization

After initial power-on or host controller reset (hardware or through USBCMD[RST]), all of the operational registers are at their default values. After a hardware reset, only the operational registers are at their default values.

To configure the external ULPI PHY the following initialization sequence is required:

1. The UTMI PHY should remain disabled if the ULPI is being used.
2. Set the CONTROL[PHY\_CLK\_SEL] bits to select the ULPI PHY as the source of USB controller PHY clock.
3. Wait for PHY clock to become valid. This can be determined by polling the CONTROL[PHY\_CLK\_VALID] status bit. Note that this bit is not valid once the CONTROL[USB\_EN] bit is set

Once the PHY clock is valid the user can proceed to the host controller initialization phase.

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 DR 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.

### 13.6.2 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.

### 13.6.3 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 DR 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 13-64](#) for summary of behavior for over-current detection when the controller is halted (suspended from a device component point of view).

### 13.6.4 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 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.

#### 13.6.4.1 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 microframe 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 microframes 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  $\mu$ sec. 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 13-64 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 13-64. 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]

<sup>1</sup> Hardware interrupt issued if USBINTR[PCE] (port change interrupt enable) is set.

<sup>2</sup> PME# asserted if enabled (Note: PME Status must always be set).

<sup>3</sup> PME# not asserted.

### 13.6.5 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 13.3.2.6, “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 13.5, “Host Data Structures.” In each microframe, 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 13-43). 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 microframe.

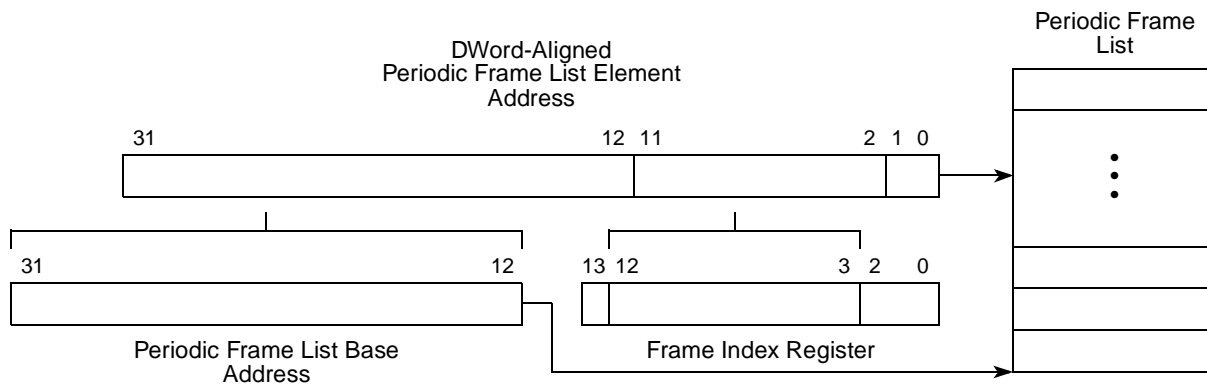
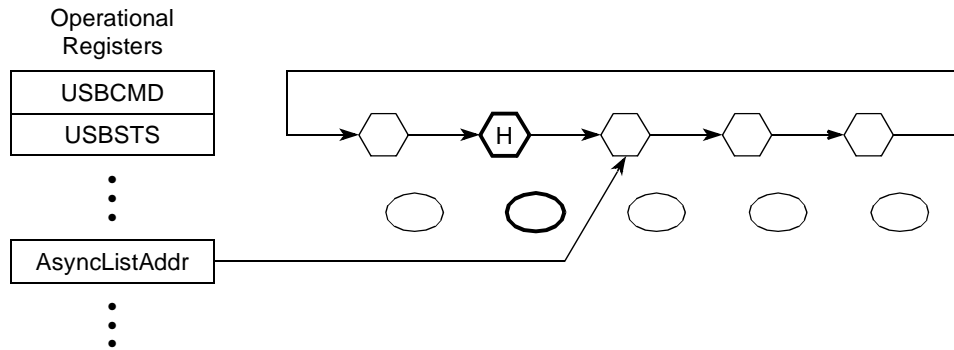


Figure 13-43. Derivation of Pointer into Frame List Array

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 13-44.

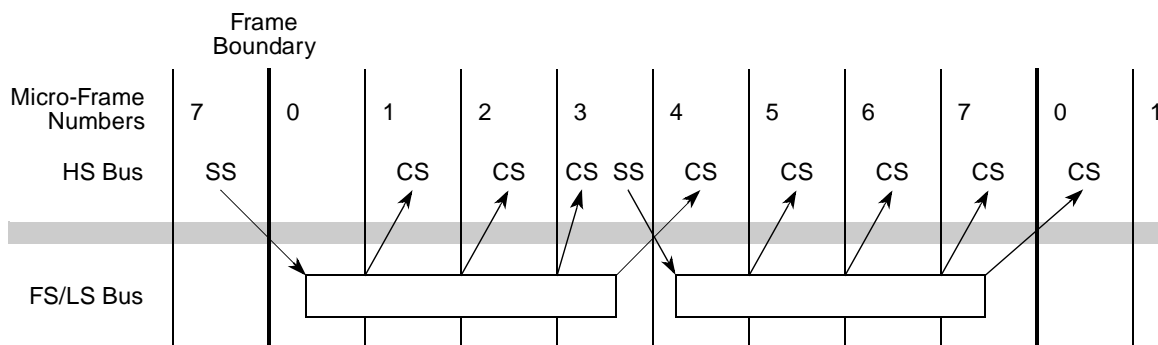


**Figure 13-44. General Format of Asynchronous Schedule List**

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.

### 13.6.6 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 microframe pipeline (see start- (SS) and complete- (CS) splits illustrated in Figure 13-45). 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.



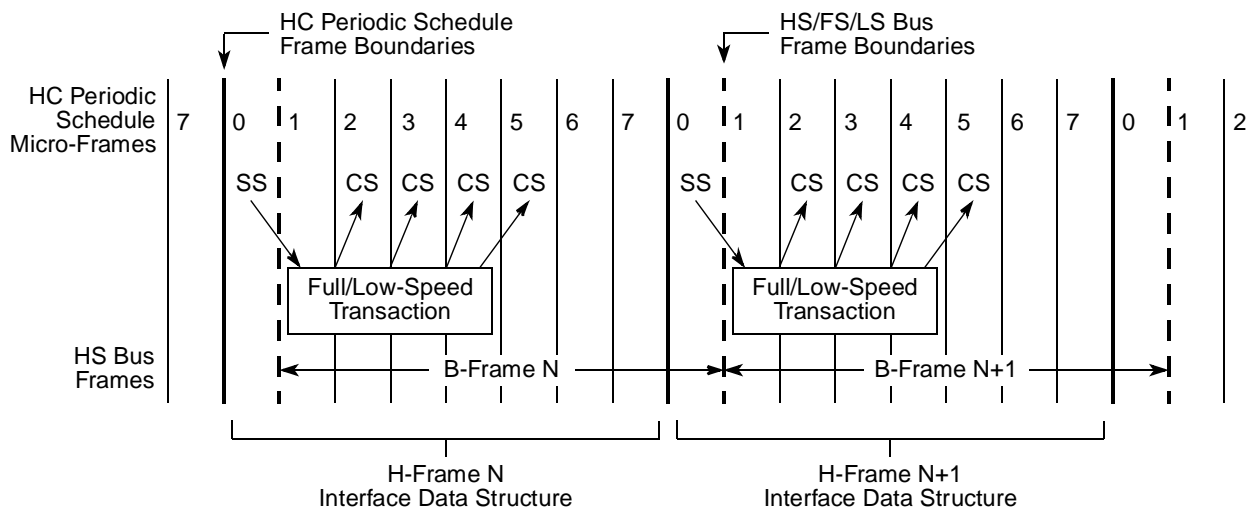
**Figure 13-45. Frame Boundary Relationship Between HS Bus and FS/LS Bus**

The simple projection, as Figure 13-45 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 microframe 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 microframe 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 microframe. The one microframe delay yields a host controller periodic schedule and bus frame boundary relationship as illustrated in Figure 13-46. 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 13-46 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 13-46. Relationship of Periodic Schedule Frame Boundaries to Bus Frame Boundaries**

H-Frame boundaries for the host controller correspond to increments of FRINDEX[13–3]. Microframe 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. Microframe 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 microframe time) illustrated in Figure 13-46. 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 13.3.2.4, “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 microframe count. Table 13-65 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 13.3.2.4, “Frame Index Register (FRINDEX),” provides the requirements that software should adhere when writing a new value in FRINDEX.

**Table 13-65. 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

### 13.6.7 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 microframe. 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 13-47 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.

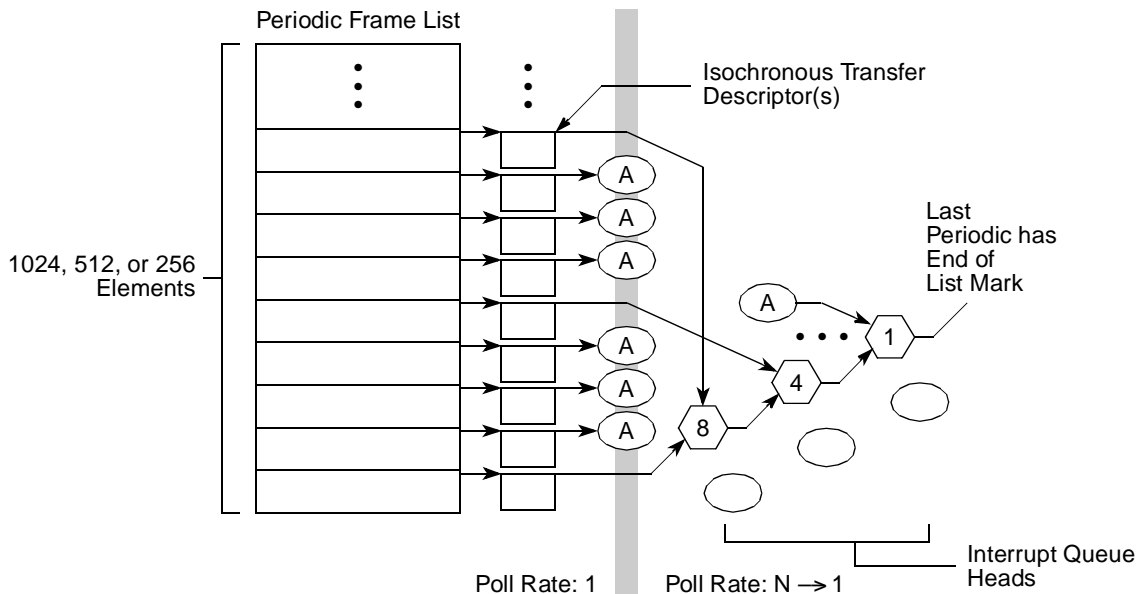


Figure 13-47. Example Periodic Schedule

### 13.6.8 Managing Isochronous Transfers Using iTDs

The structure of an iTD is presented in isochronous (high-speed) transfer descriptor (iTID). There are four distinct sections to an iTD:

- Next link pointer
  - This is the first field.
  - This field is for schedule linkage purposes only.
- Transaction description array
  - This is an eight-element array.
  - Each element represents control and status information for one microframe's worth of transactions for a single high-speed isochronous endpoint.
- Buffer page pointer array
  - This 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.
  - Its fields are used across all transactions executed for this iTD, including endpoint addressing, transfer direction, maximum packet size and high-bandwidth multiplier.

### 13.6.8.1 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 microframes 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 than 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 microframe. In other words, the Mult field represents a transaction count for the endpoint in the current microframe. 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 microframe. 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 its 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  $3 \times 1024$  bytes.

For IN transfers, the host controller issues Mult transactions. It is assumed that software has properly initialized the iTD to accommodate all 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 microframe, Transaction  $n$  Length contains the total bytes received. The following actions can occur:

- If the final value of Transaction  $n$  Length is less than the value of Maximum Packet Size, less data was received than was allowed for from the associated endpoint. This short packet condition does not set USBSTS[UI] (USB interrupt). The host controller does not detect this condition.
- If the device sends more than Transaction  $n$  Length or Maximum Packet Size bytes (whichever is less), the host controller sets the Babble Detected bit and clears 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, the host controller automatically executes the value of Mult transactions. The host controller does not execute all Mult transactions in the following cases:
  - The endpoint is an OUT and Transaction  $n$  Length goes to zero before all the Mult transactions have executed (ran out of data).
  - 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 microframe 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 is written back to the iTD; and the host controller proceeds to processing the next microframe.

### 13.6.8.2 Software Operational Model for iTDs

A client buffer request to an isochronous endpoint may span 1 to N microframes. 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 13-48 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).

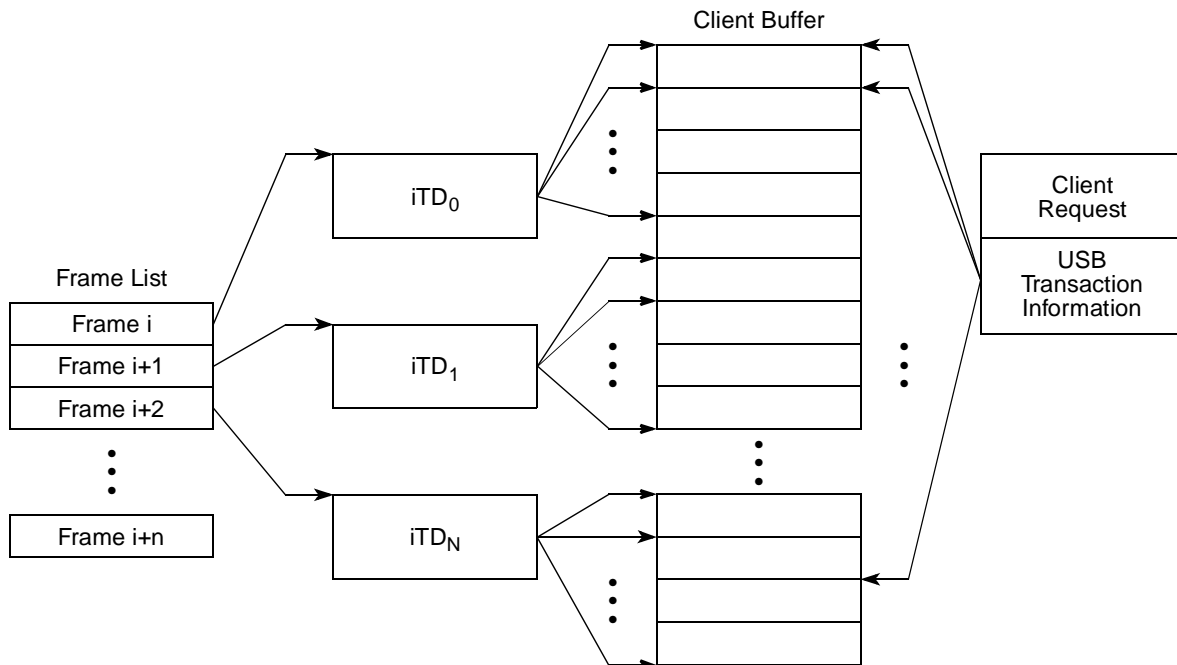


Figure 13-48. Example Association of iTDs to Client Request Buffer

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 microframe's worth of transactions. The EHCI controller does not provide per transaction results within a microframe. It treats the per microframe 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, 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.

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.

### 13.6.8.2.1 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 microframes 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 microframes. The three caching models are: no caching, microframe 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 microframe the host controller is currently executing. Of course, there is no information about where in the microframe the host controller is, so a constant uncertainty factor of one microframe 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 microframe) but will always dump any accumulated schedule state at the end of the microframe. At the appropriate time relative to the beginning of every microframe, 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 microframes 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 microframes). Software uses the value of the FRINDEX register (plus the constant 1 uncertainty) to determine the current microframe/frame (assume modulo 8 arithmetic in adding the constant 1 to the microframe number). For any current frame  $N$ , if the current microframe is 0 to 6, then software can safely add isochronous transactions to Frame  $N + 1$ . If the current microframe is 7, then software can add isochronous transactions to Frame  $N + 2$ .

Microframe 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 microframes indicated in the Isochronous Scheduling Threshold field. For example, if the count value were 2, then the host controller keeps a window of two microframes worth of

state (current microframe, plus the next) on chip. On each microframe boundary, the host controller releases the current microframe state and begins accumulating the next microframe state.

### 13.6.9 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 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 microframe 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 13-44](#). Queue head data structures are the only valid data structures that may be linked into the asynchronous schedule. An isochronous transfer descriptor (iTID 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.

### 13.6.9.1 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

```

### 13.6.9.2 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.

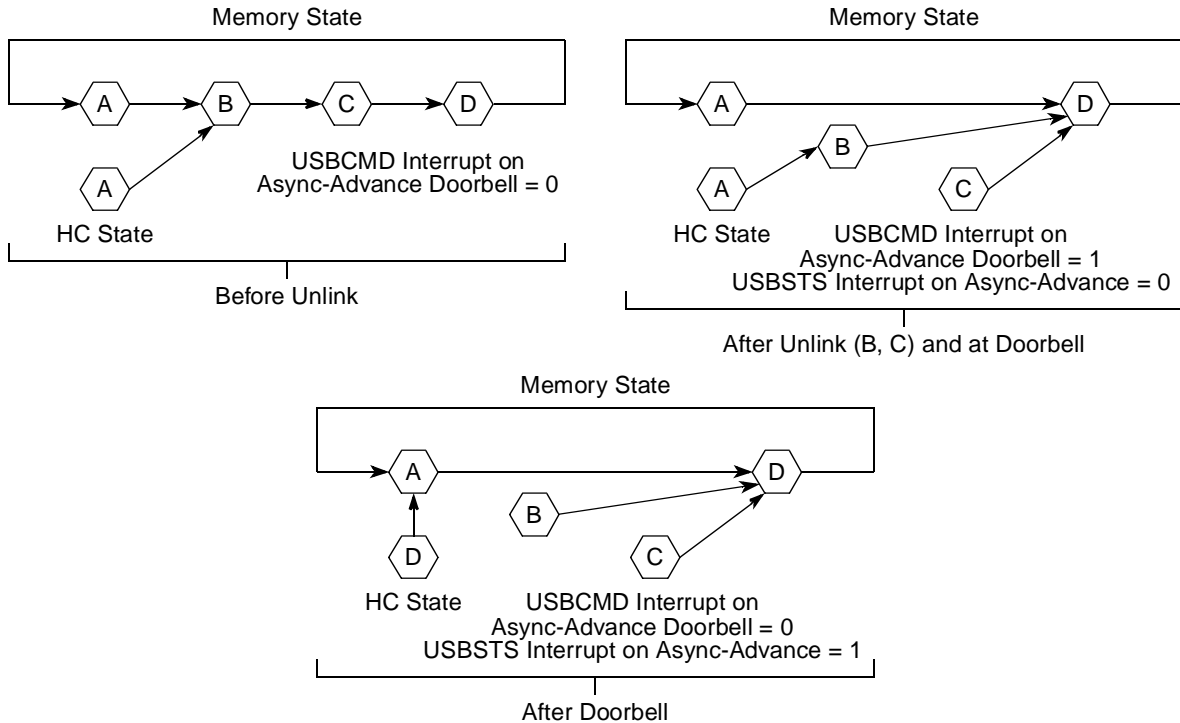
The method software uses to determine when it is safe to modify a removed queue head is to handshake with the host controller. The handshake mechanism allows software to remove items from the asynchronous schedule, then execute a simple, lightweight handshake that is used by software as a key that it 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 13-49** 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).



**Figure 13-49. 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

### 13.6.9.3 Empty Asynchronous Schedule Detection

EHCI uses two bits to detect when the asynchronous schedule is empty. The queue head data structure (see [Figure 13-41](#)) 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 13.6.9.4, “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.

Figure 13-50 shows an example illustrating the H-bit in a schedule.

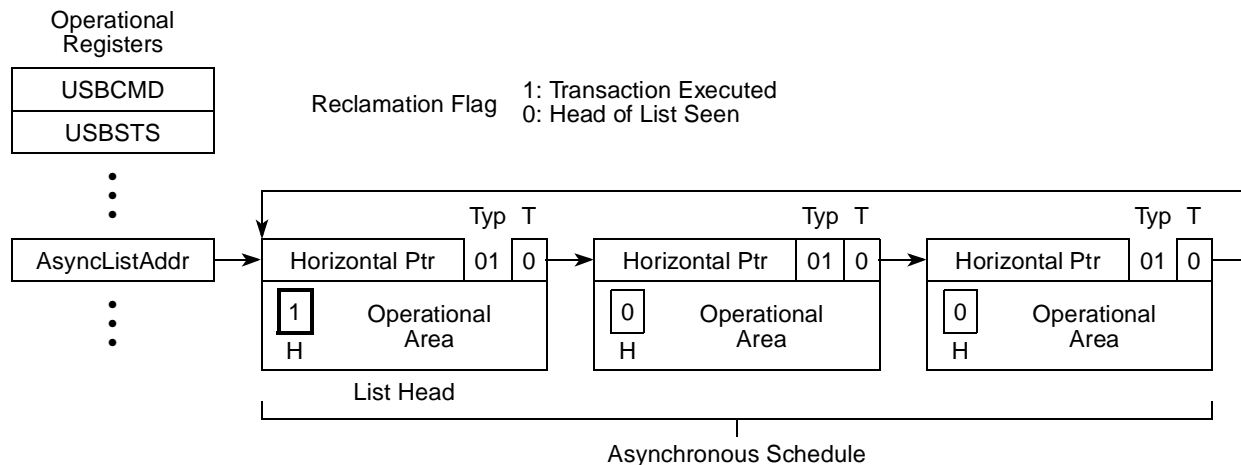


Figure 13-50. Asynchronous Schedule List with Annotation to Mark Head of List

#### 13.6.9.4 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 microframe. In addition, it may have idled itself early in a microframe. When this occurs (idles early in the microframe) the host controller must occasionally reactivate during the microframe 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 microframe is equivalent to the transition from the periodic schedule, or
- The asynchronous schedule traversal restarts from a sleeping state.

#### 13.6.9.5 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.

#### 13.6.10 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 13.5.5, “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.

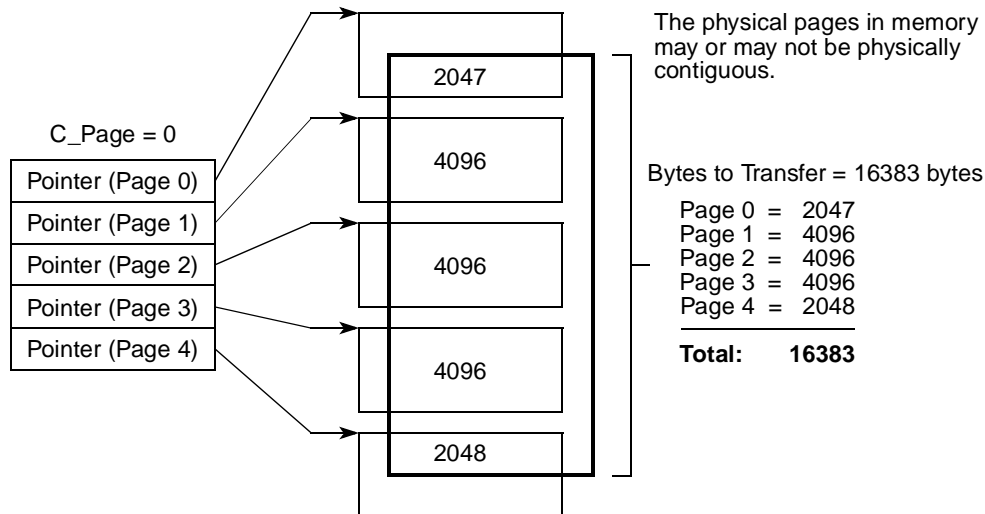
### 13.6.10.1 Buffer Pointer List Use for Data Streaming with qTDs

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 13-51 illustrates these requirements.



**Figure 13-51. 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 13-51 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.

### 13.6.10.2 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 microframe 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 13-66](#).

**Table 13-66. Example Periodic Reference Patterns for Interrupt Transfers**

Frame # Reference Sequence	Description
0, 2, 4, 6, 8, .... S-Mask = 0x01	A queue head for the interval of 2 milliseconds (16 microframes) 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 microframe 0 of the frame.
0, 2, 4, 6, 8, ... S-Mask = 0x02	Another example of a queue head with a interval 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 microframe 1 of the frame.

### 13.6.10.3 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.

## 13.6.11 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 13-55](#)). 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 13-67](#) 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 13-67. Ping Control State Transition Table**

Current	Event		Next
	Host	Device	
Do Ping	PING	Nak	Do Ping
Do Ping	PING	Ack	Do OUT
Do Ping	PING	XactErr <sup>1</sup>	Do Ping
Do Ping	PING	Stall	N/C <sup>2</sup>
Do OUT	OUT	Nak	Do Ping
Do OUT	OUT	Nyet	Do Ping <sup>3</sup>
Do OUT	OUT	Ack	Do OUT
Do OUT	OUT	XactErr <sup>1</sup>	Do Ping
Do OUT	OUT	Stall	N/C <sup>2</sup>

<sup>1</sup> Transaction Error (XactErr) is any time the host misses the handshake.

<sup>2</sup> 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.

<sup>3</sup> 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 13-55](#). The defined ping protocol allows the host to be imprecise on the initialization of the ping protocol (that is, start in Do OUT when there is no information 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.

### 13.6.12 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.

#### 13.6.12.1 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.

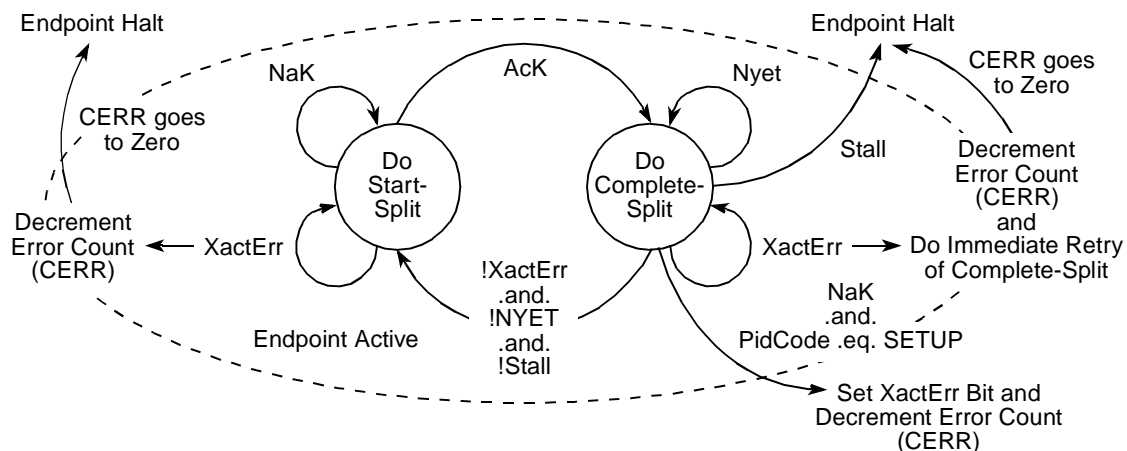


Figure 13-52. Host Controller Asynchronous Schedule Split-Transaction State Machine

### 13.6.12.1.1 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.

### 13.6.12.1.2 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 microframe 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 microframe, 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.

### 13.6.12.2 Split Transaction Interrupt

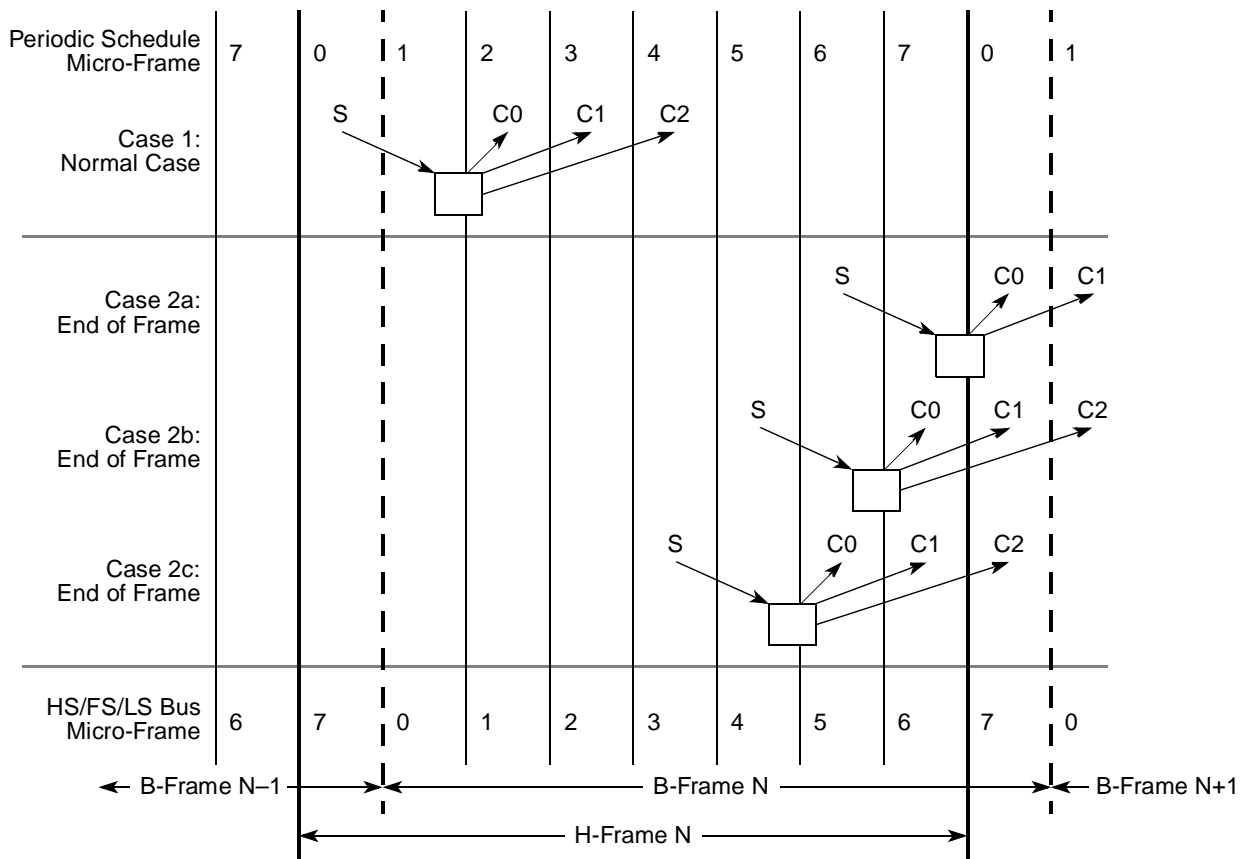
Split-transaction Interrupt-IN/OUT endpoints are managed using the same data structures used for high-speed interrupt endpoints. They both co-exist in the periodic schedule. Queue heads/qTDs offer the set of features required for reliable data delivery, which is characteristic to interrupt transfer types. 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.

#### 13.6.12.2.1 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 microframes 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 microframes, or the data or response information in the pipeline is lost. [Figure 13-53](#) illustrates the general scheduling boundary conditions that are supported by the EHCI periodic schedule

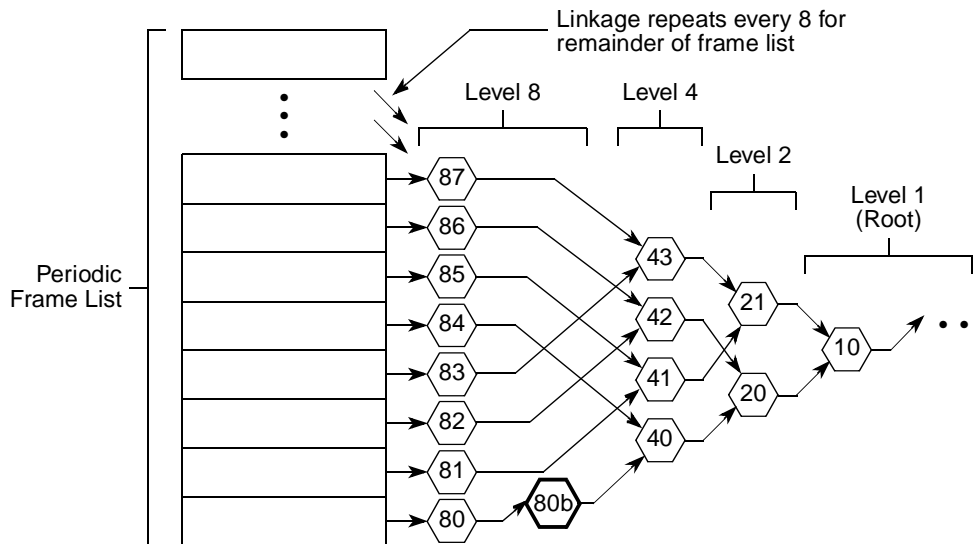
and queue head data structure. The S and  $C_n$  labels indicate microframes where software can schedule start-splits and complete splits (respectively).



**Figure 13-53. 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 microframe 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 13-54](#) illustrates the general layout of the periodic schedule.



**Figure 13-54. 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 13.5.7, “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 13-55](#)). 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 microframe (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 13-53](#), 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 microframe 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 microframes (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 13-53](#), 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 microframe 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.

### 13.6.12.2.2 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 13.5.7](#), “[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 microframes 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 microframes 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 microframe.

The rules for schedule traversal and limited execution while in Recovery Path mode are:

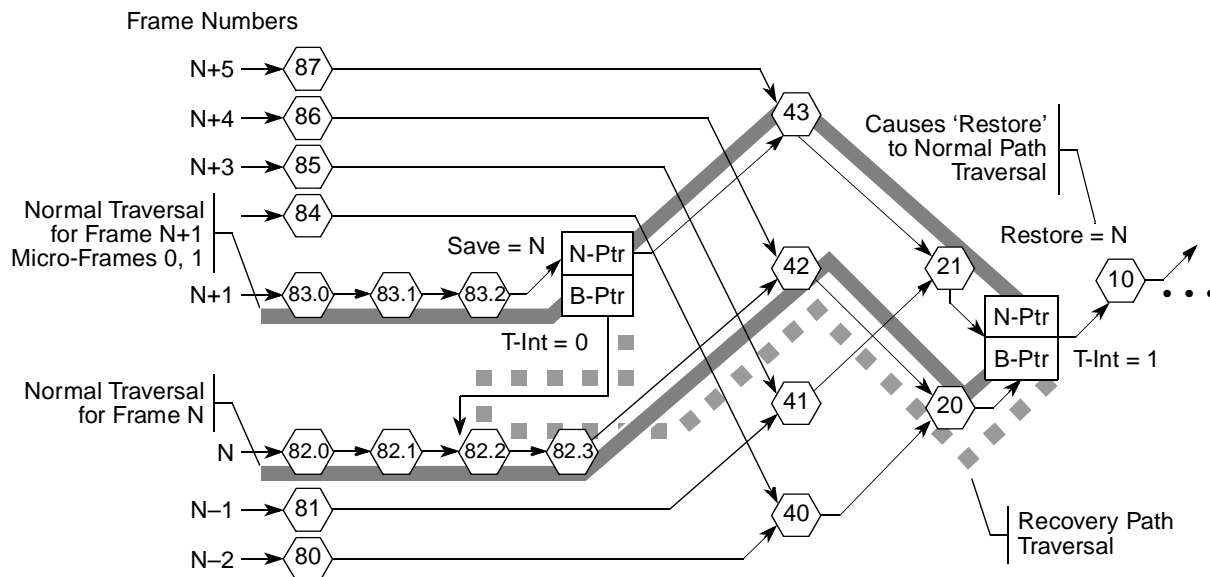
- Always follow the Normal Path Link Pointer when it encounters an FSTN that is a Save-Place indicator. The host controller must not recursively follow Save-Place FSTNs. Therefore, while executing in Recovery Path mode, it must never follow an FSTN's Back Path Link Pointer.
- Do not process an siTD or 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 microframe 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 microframe, the host controller starts traversal at the frame list.

An example traversal of a periodic schedule that includes FSTNs is illustrated in [Figure 13-55](#).



**Figure 13-55. Example Host Controller Traversal of Recovery Path via FSTNs**

In frame N (microframes 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 microframes 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: {82.0, 82.1, 82.2, 82.3, 42, 20, Restore-N, 10 ...}.

In frame N+1 (microframes 0 and 1), when the host controller encounters Save-Place FSTN (Save-N), it observes that Save-N.Back Path Link Pointer.T-bit is zero (definition of a Save-Place 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 13-55](#) 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 microframes include: {8<sub>3,0</sub>, 8<sub>3,1</sub>, 8<sub>3,2</sub>, Save-A, 8<sub>2,2</sub>, 8<sub>2,3</sub>, 4<sub>2</sub>, 2<sub>0</sub>, Restore-N, 4<sub>3</sub>, 2<sub>1</sub>, Restore-N, 10 ...}.

In frame N+1 (microframes 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 microframes include: {8<sub>3,0</sub>, 8<sub>3,1</sub>, 8<sub>3,2</sub>, Save-A, 4<sub>3</sub>, 2<sub>1</sub>, Restore-N, 1<sub>0</sub> ...}.

### 13.6.12.2.3 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 is used 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 are 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.

### 13.6.12.2.4 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 microframe 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).

### 13.6.12.2.5 Split Transaction Execution State Machine for Interrupt

In the following section, all references to microframe are in the context of a microframe 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 microframe number. It is an eight-bit value calculated by the host controller at the beginning of every microframe. It is calculated from the three least significant bits of the FRINDEX register (that is,  $cMicroFrameBit = (1 \text{ shifted-left}(FRINDEX[2-0]))$ ). The cMicroFrameBit has at most one bit asserted, which always

corresponds to the current microframe number. For example, if the current microframe 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 microframe.

Figure 13-56 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 is 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).

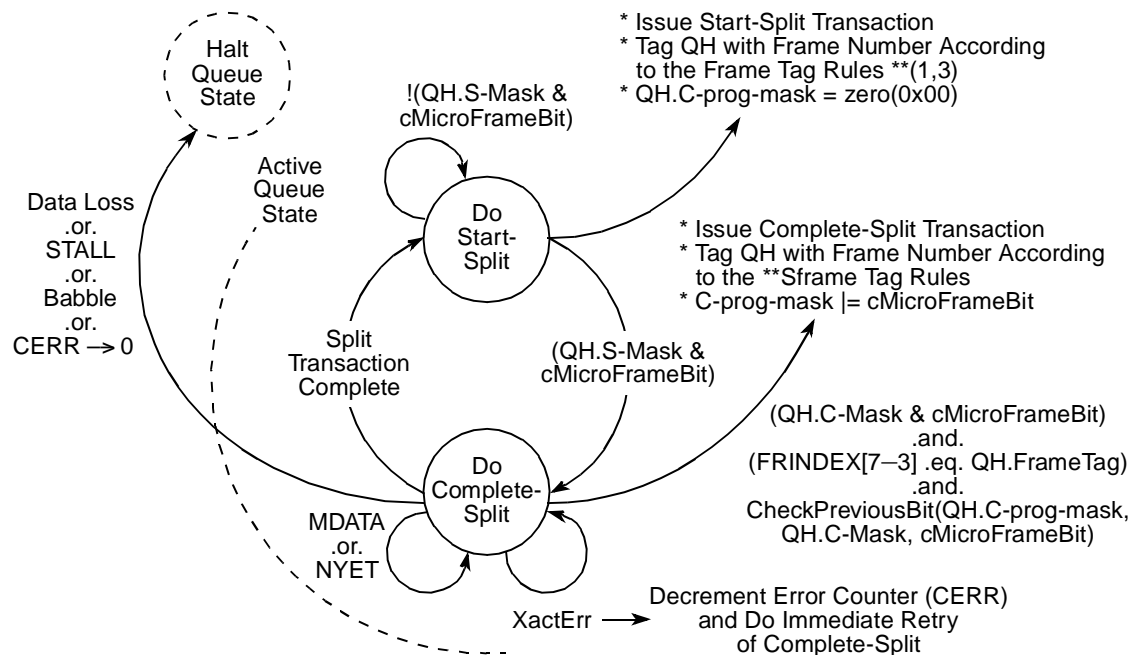


Figure 13-56. Split Transaction State Machine for Interrupt

### 13.6.12.2.6 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 13.6.12.2.7, “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.

### 13.6.12.2.7 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 microframe.
- **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 microframe. 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 microframe. 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 microframe. 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 microframe 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 microframe 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 13-68](#) lists the possible combinations and the appropriate action.

**Table 13-68. Interrupt IN/OUT Do Complete Split State Execution Criteria**

Condition	Action	Description
not(A) not(D)	Ignore QHD	Neither a start nor complete-split is scheduled for the current microframe. Host controller should continue walking the schedule.
A not(C)	If PIDCode = IN Halt QHD If PIDCode = OUT Retry start-split	Progress bit check failed. This 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 Microframe bit in the status field to a one.
A not(B) C	If PIDCode = IN Halt QHD If PIDCode = OUT 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 Microframe bit in the status field to a one.
A B C not(D)	Execute complete-split	This is the non-error case where the host controller executes a complete-split transaction.
D	If PIDCode = IN Halt QHD If PIDCode = OUT 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 Microframe 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.

### 13.6.12.2.8 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 13-53](#)).
- 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 13-53](#).

- 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 13-53](#).

### 13.6.12.2.9 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 microframe, 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 its 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.

### 13.6.12.3 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 13.6.8, “Managing Isochronous Transfers Using iTDs,”](#) for the operational model of iTDs) with the contiguous data feature provided by queue heads. This simple arrangement allows

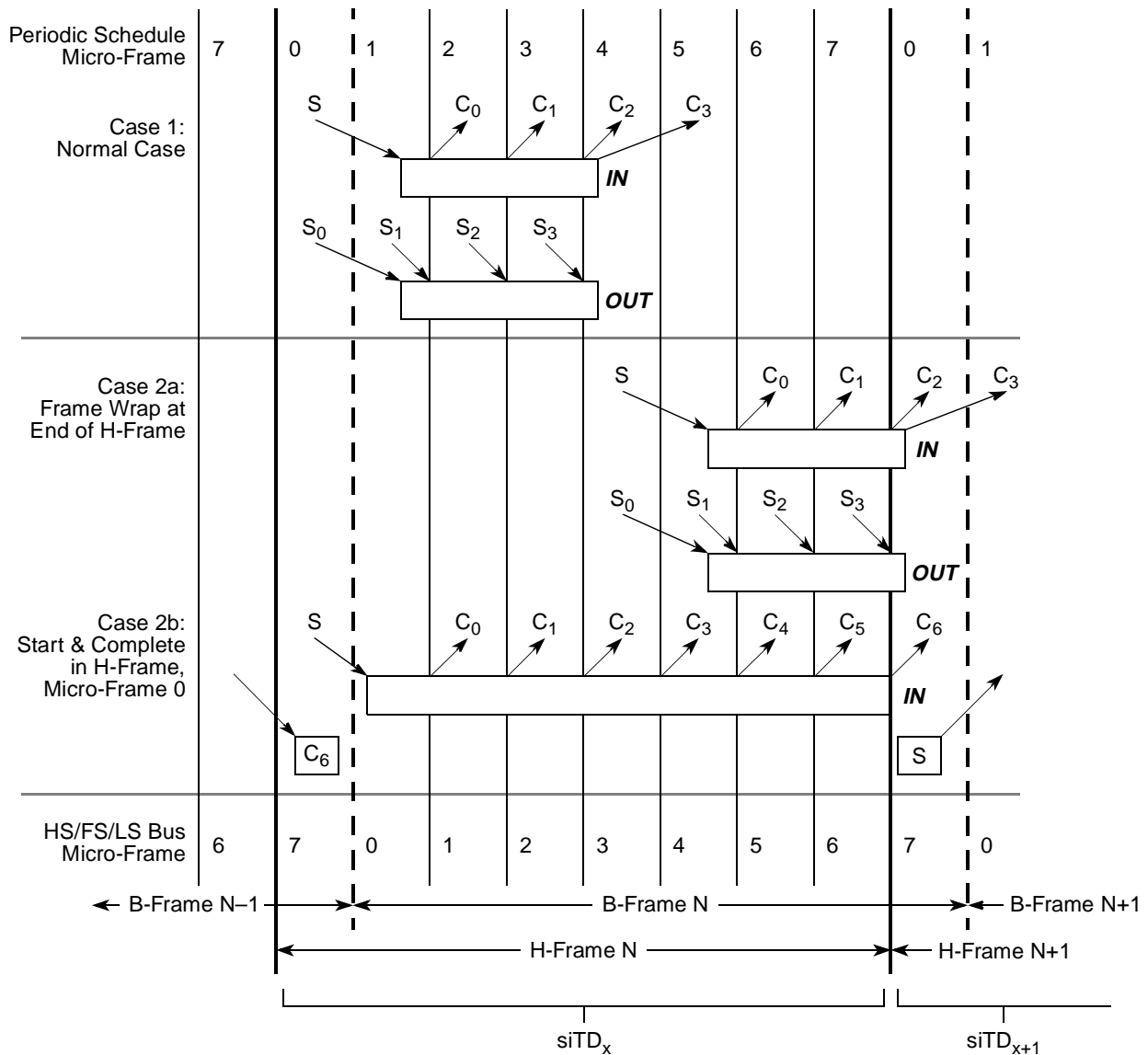
a single isochronous scheduling model and adds the additional feature that all data received from the endpoint (per split transaction) must land into a contiguous buffer.

### 13.6.12.3.1 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 microframes the start-splits and complete-splits for each full-speed isochronous endpoint occur. The requirements described in [Section 13.6.12.2.1, “Split Transaction Scheduling Mechanisms for Interrupt,”](#) apply.

[Figure 13-57](#) illustrates the general scheduling boundary conditions that are supported by the EHCI periodic schedule. The  $S_n$  and  $C_n$  labels indicate microframes 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 13-57 illustrates the relationship of an siTD to the H-Frame.



**Figure 13-57. 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, microframes 6 or 7 (H-Frame microframe 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 microframe. 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 13-49](#)). This bit is used to track the current state of the split transaction. The rules for managing this bit are described in [Section 13.6.12.3.3, “Split Transaction Execution State Machine for Isochronous.”](#)
- **Frame S-mask**  
This is a bit-field wherein system software sets a bit corresponding to the microframe (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 13-57](#), 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 microframe 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 microframes (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 13-57](#), 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 microframe 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 and 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 13-58 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.

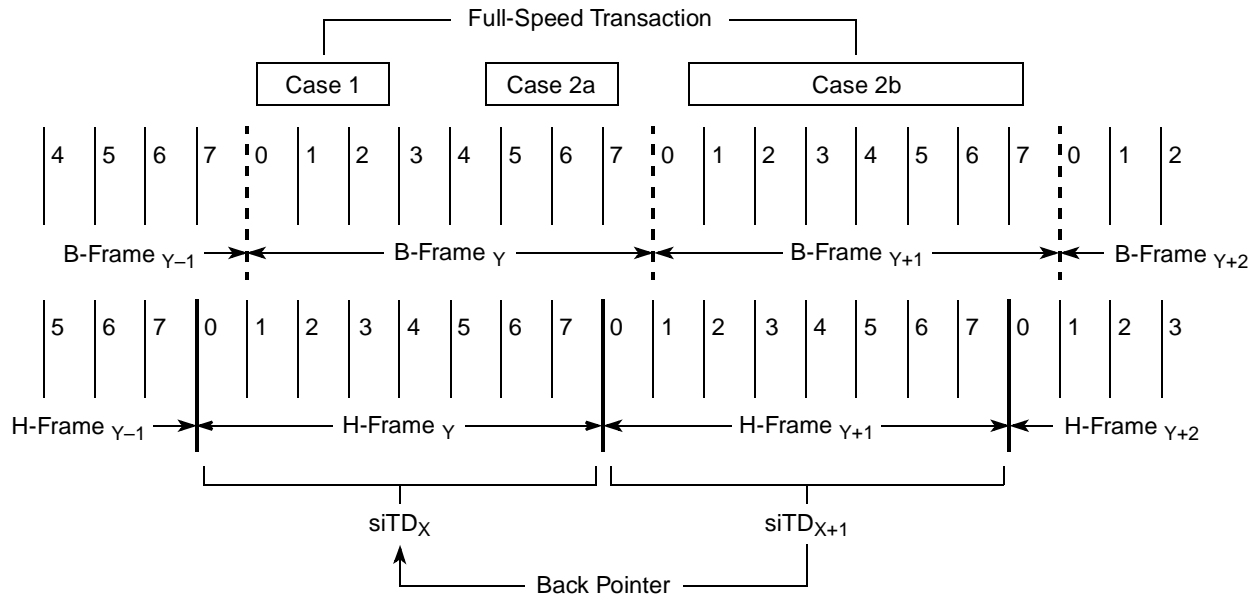


Figure 13-58. siTD Scheduling Boundary Examples

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. siTD<sub>X</sub> 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 microframe 7 of H-Frame <sub>$\gamma+1$</sub> , or microframe 0 of H-Frame <sub>$\gamma+2$</sub> . The complete splits are scheduled using siTD <sub>$X+2$</sub>  (not shown). The complete-splits to extract this data must use the buffer pointer from siTD <sub>$X+1$</sub> . The only way for the host controller to reach siTD <sub>$X+1$</sub>  from H-Frame <sub>$\gamma+2$</sub>  is to use siTD <sub>$X+2$</sub> '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, microframe 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 microframe 1 of H-Frame N and the last complete-split would need to occur in microframe 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.

### 13.6.12.3.2 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 microframes), timeouts and corrupted data received.

In similar kind to interrupt split-transactions, the portions of the split transaction protocol must execute in the microframes 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 microframe 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 microframe).

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 13.6.12.3.1, “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 microframe (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 microframe 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 microframe boundary, the transaction translator could have received the entire packet (including both CRC bytes) but not received the packet EOP. In the next microframe, 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 its 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.

### 13.6.12.3.3 Split Transaction Execution State Machine for Isochronous

In this section, all references to microframe are in the context of a microframe 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 13.6.12.3.2, “Tracking Split Transaction Progress for Isochronous Transfers,” plus the variable `cMicroFrameBit` defined in Section 13.6.12.2.5, “Split Transaction Execution State Machine for Interrupt,” to track the progress of an isochronous split transaction. Figure 13-59 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.

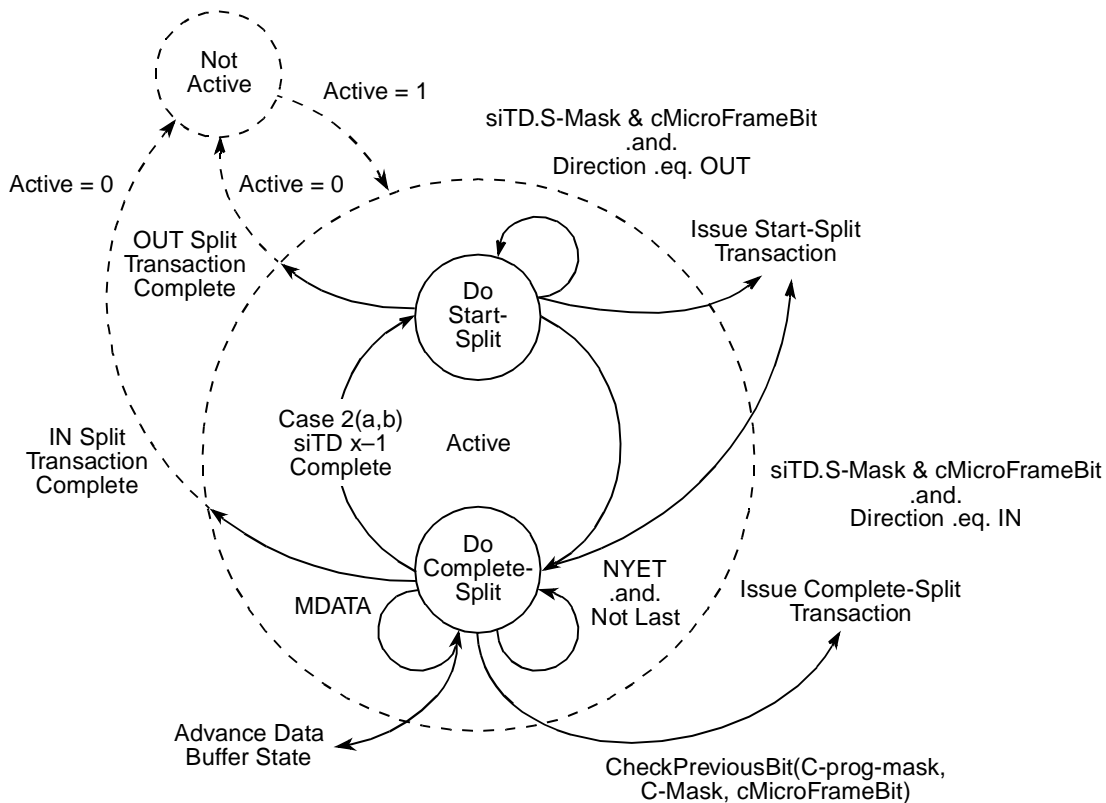


Figure 13-59. Split Transaction State Machine for Isochronous

#### 13.6.12.3.4 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 13-58](#)) is used to determine the initial value of TP. The initial cases are summarized in [Table 13-69](#).

**Table 13-69. 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 13-70](#) illustrates all of the TP and T-count transitions, which must be accomplished by the host controller.

**Table 13-70. 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 13-70](#).

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, the host controller issues 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 yields 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 does 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 13.6.12.3.7, “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 13.6.12.3.5, “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 microframes. 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.

### 13.6.12.3.5 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 sequence in which they are applied depends on which microframe 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. The individual tests are as follows.

- 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 microframe. 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 13.6.12.2.7, “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, this is a case 2a or 2b scheduling boundary (see [Figure 13-57](#)). See [Section 13.6.12.3.6, “Complete-Split for Scheduling Boundary Cases 2a, 2b,”](#) for details in handling this condition.

If Test A and Test B evaluate to true, 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 perform the following actions:

1. Decrement the number of bytes received from siTD[Total Bytes To Transfer]
2. Adjust siTD[Current Offset] by the number of bytes received
3. Adjust the siTD[P] (page select) field if the transfer caused the host controller to use the next page pointer
4. 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 microframe occurs, the Active bit is cleared.

- DATA<sub>x</sub> (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 DATA<sub>x</sub> 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 13.6.12.2.7, “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 microframe X to X+1 and during microframe X, the transaction translator responds with an MDATA and the data accumulated up to the end of microframe 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.

### 13.6.12.3.6 Complete-Split for Scheduling Boundary Cases 2a, 2b

Boundary cases 2a and 2b (INs only) (see [Figure 13-57](#)) require that the host controller use the transaction state context of the previous siTD to finish the split transaction. [Table 13-71](#) enumerates the transaction state fields.

**Table 13-71. Summary siTD Split Transaction State**

Buffer State	Status	Execution Progress
Total Bytes To Transfer P (page select) Current Offset TP (transaction position) 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 siTD<sub>X</sub>[Back Pointer] T-bit is zero, or
- If cMicroFrameBit is a 0x2 and siTD<sub>X</sub>[S-mask[0]] is zero

When either of these conditions apply, then the host controller must use the transaction state from siTD<sub>X-1</sub>.

In order to access siTD<sub>X-1</sub>, the host controller reads on-chip the siTD referenced from siTD<sub>X</sub>[Back Pointer].

The host controller must save the entire state from siTD<sub>X</sub> while processing siTD<sub>X-1</sub>. This is to accommodate for case 2b processing. The host controller must not recursively walk the list of siTD[Back Pointers].

If siTD<sub>X-1</sub> 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 13-71](#)) of siTD<sub>X-1</sub> is appropriately advanced based on the results and written back to memory. If the resultant state of siTD<sub>X-1</sub>'s Active bit is a one, then the host controller returns to the context of siTD<sub>X</sub>, and follows its next pointer to the next schedule item. No updates to siTD<sub>X</sub> are necessary.

If siTD<sub>X-1</sub> 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 siTD<sub>X-1</sub>'s Active bit is cleared, (because it was cleared when the host controller first visited siTD<sub>X-1</sub> via siTD<sub>X</sub>'s back pointer, it transitioned to zero as a result of a detected error, or the results of siTD<sub>X-1</sub>'s complete-split transaction cleared it), then the host controller returns to the context of siTD<sub>X</sub> 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 siTD<sub>X</sub>[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.

### 13.6.12.3.7 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 13-72](#) illustrates a few frames worth of scheduling required to schedule a case 2a full-speed isochronous data stream.

**Table 13-72. Example Case 2a—Software Scheduling  $siTD$ s for an IN Endpoint**

$siTD_X$		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  $siTD$ s for the transaction stream. Since this is the case-2a frame-wrap case, S-masks of all  $siTD$ s for this endpoint have a value of 0x10 (a one bit in microframe 4) and C-mask value of 0xC3 (one-bits in microframes 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 microframes 0 and 1 are ignored because the state is Do Start Split. During microframe 4, the host controller determines that it can run a start-split (and does) and changes SplitXState to Do Complete Split. During microframes 6 and 7, the host controller executes complete-splits. Notice the siTD for frame X+1 has its 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 microframes 0 and 1 it will detect that it must execute complete-splits.

During H-Frame X+1, microframe 0, the host controller detects that siTD<sub>X+1</sub>'s Back Pointer[T] bit is a zero, saves the state of siTD<sub>X+1</sub> and fetches siTD<sub>X</sub>. It executes the complete split transaction using the transaction state of siTD<sub>X</sub>. If the siTD<sub>X</sub> split transaction is complete, siTD's Active bit is cleared and results written back to siTD<sub>X</sub>. The host controller retains the fact that siTD<sub>X</sub> is retired and transitions the SplitXState in siTD<sub>X+1</sub> to Do Start Split. At this point, the host controller is prepared to execute the start-split for siTD<sub>X+1</sub> when it reaches microframe 4. If the split-transaction completes early (transaction-complete is defined in [Section 13.6.12.3.5, “Periodic Isochronous—Do Complete Split”](#)), that is, before all the scheduled complete-splits have been executed, the host controller changes siTD<sub>X</sub>[SplitXState] to Do Start Split early and naturally skips the remaining scheduled complete-split transactions. For this example, siTD<sub>X+1</sub> does not receive a DATA0 response until H-Frame X+2, microframe 1.

During H-Frame X+2, microframe 0, the host controller detects that siTD<sub>X+2</sub>'s Back Pointer[T] bit is zero, saves the state of siTD<sub>X+2</sub> and fetches siTD<sub>X+1</sub>. 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<sub>X+2</sub>, and traverses its next pointer without any state change updates to siTD<sub>X+2</sub>.

During H-Frame X+2, microframe 1, the host controller detects siTD<sub>X+2</sub>'s S-mask[0] bit is zero, saves the state of siTD<sub>X+2</sub> and fetches siTD<sub>X+1</sub>. 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<sub>X+2</sub> and changes its SplitXState to Do Start Split. At this point, the host controller is prepared to execute start-splits for siTD<sub>X+2</sub> when it reaches microframe 4.

### 13.6.13 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, assuming the CF-bit in the CONFIGFLAG register is set, is as follows:

1. Disable the periodic and asynchronous schedules by clearing the USBCMD[ASE] and USBCMD[PSE].
2. Place all enabled root ports into the suspended state by setting the Suspend bit in the PORTSC register (PORTSC[SUSP]).
3. 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.



4. 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.
5. 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].

### 13.6.14 Interrupts

The EHCI host controller hardware provides interrupt capability based on a number of sources. The following list describes the 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.)
- 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 microframes. This means that the host controller will not generate interrupts any more frequently than once every eight microframes.

[Section 13.6.14.2.4, “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 read 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.

**13.6.14.1 Transfer/Transaction Based Interrupts**

These interrupt sources are associated with transfer and transaction progress. They are all dependent on the next interrupt threshold.

**13.6.14.1.1 Transaction Error**

A transaction error is any error that caused the host controller to think that the transfer did not complete successfully. [Table 13-73](#) 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 13-73. Summary of Transaction Errors**

Event/ Result	Queue Head/qTD/iTD/siTD Side Effects		USBSTS[USBERRINT]
	Cerr	Status Field	
CRC	-1	XactErr set	1 <sup>1</sup>
Timeout	-1	XactErr set	1 <sup>1</sup>
Bad PID <sup>2</sup>	-1	XactErr set	1 <sup>1</sup>
Babble	N/A	See <a href="#">Section 13.6.14.1.2, "Serial Bus Babble"</a>	1
Buffer Error	N/A	See <a href="#">Section 13.6.14.1.3, "Data Buffer Error"</a>	

<sup>1</sup> 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.

<sup>2</sup> 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.

**13.6.14.1.2 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 microframe 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.

#### 13.6.14.1.3 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*.

### 13.6.14.1.4 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.

### 13.6.14.1.5 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.

## 13.6.14.2 Host Controller Event Interrupts

These interrupt sources are independent of the interrupt threshold (with the one exception being the Interrupt on Async Advance).

### 13.6.14.2.1 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)

### 13.6.14.2.2 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.

### 13.6.14.2.3 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 13.6.9.2, “Removing Queue Heads from Asynchronous Schedule.”](#)

### 13.6.14.2.4 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 13-74 summarizes the required actions taken on the various host errors.

**Table 13-74. 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.

## 13.7 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 DR 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.

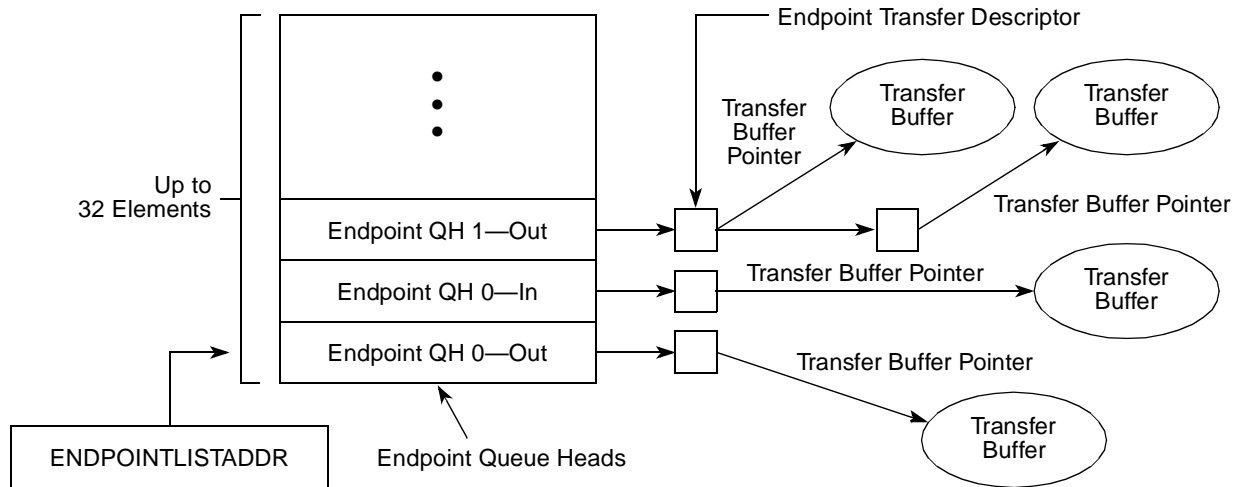


Figure 13-60. Endpoint Queue Head Organization

### 13.7.1 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 13-61 shows the Endpoint Queue Head structure.

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
Mult		zlt		00		Maximum Packet Length										ios		000_0000_0000_0000										0x00				
Current dTD Pointer <sup>1</sup>															0_0000		0x04															
Next dTD Pointer <sup>1</sup>															0000		T <sup>1</sup>	0x08 <sup>2</sup>														
0	Total Bytes <sup>1</sup>										ioc <sup>1</sup>		000		MultO <sup>1</sup>		00		Status <sup>1</sup>				0x0C <sup>2</sup>									
Buffer Pointer (Page 0) <sup>1</sup>										Current Offset <sup>1</sup>								0x10 <sup>2</sup>														
Buffer Pointer (Page 1) <sup>1</sup>										Reserved								0x14 <sup>2</sup>														
Buffer Pointer (Page 2) <sup>1</sup>										Reserved								0x18 <sup>2</sup>														
Buffer Pointer (Page 3) <sup>1</sup>										Reserved								0x1C <sup>2</sup>														
Buffer Pointer (Page 4) <sup>1</sup>										Reserved								0x20 <sup>2</sup>														
Reserved																0x24																
Setup Buffer Bytes 3–0 <sup>1</sup>																0x28																
Setup Buffer Bytes 7–4 <sup>1</sup>																0x2C																

**Figure 13-61. Endpoint Queue Head Layout**

<sup>1</sup> Device controller read/write; all others read-only.

<sup>2</sup> Offsets 0x08 through 0x20 contain the transfer overlay.

### 13.7.1.1 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 13-75 describes the endpoint capabilities and characteristics fields.

**Table 13-75. 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: 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) 01 Execute 1 Transaction. 10 Execute 2 Transactions. 11 Execute 3 Transactions. <b>Note:</b> Non-ISO endpoints must set Mult = 00. <b>Note:</b> 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 the total transfer length is a multiple. This bit is not relevant for Isochronous transfers. 0 Enable zero length packet to terminate transfers equal to a multiple of the Maximum Packet Length. (default). 1 Disable the zero length packet on transfers that are equal in length to a multiple Maximum Packet Length.

**Table 13-75. Endpoint Capabilities/Characteristics (continued)**

Bits	Name	Description
28–27	—	Reserved, should be cleared. These bit reserved for future use and should be cleared.
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.

### 13.7.1.2 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 is 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.

### 13.7.1.3 Current dTD Pointer

The current dTD pointer is used by the device controller to locate the transfer in progress. This word is for USB\_DR (hardware) use only and should not be modified by DCD software.

Table 13-76 describes the current dTD pointer fields.

**Table 13-76. 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 is 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.

### 13.7.1.4 Setup Buffer

The setup buffer is dedicated storage for the 8-byte data that follows a setup 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 13-77 describes the multiple mode control fields.

**Table 13-77. 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.

## 13.7.2 Endpoint Transfer Descriptor (dTD)

The dTD describes the location and quantity of data to be sent/received for given transfer to the device controller. 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 13.8.5, “Managing Transfers with Transfer Descriptors.”

Figure 13-62 shows the endpoint transfer descriptor.

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
Next Link Pointer																0000		T	0x00													
0	Total Bytes <sup>1</sup>										ioc	000	MultO	00	Status <sup>1</sup>				0x04													
Buffer Pointer (Page 0)										Current Offset <sup>1</sup>							0x08															
Buffer Pointer (Page 1)										0	Frame Number <sup>1</sup>							0x0C														
Buffer Pointer (Page 2)										0000_0000_0000							0x10															
Buffer Pointer (Page 3)										0000_0000_0000							0x14															
Buffer Pointer (Page 4)										0000_0000_0000							0x18															

**Figure 13-62. Endpoint Transfer Descriptor (dTD)**

<sup>1</sup> Device controller read/write; all others read-only.

Table 13-78 describes the next dTD pointer fields.

**Table 13-78. 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 13-79 describes the next dTD token fields.

**Table 13-79. dTD Token**

Bits	Description												
31	Reserved, should be cleared. Bit reserved for future use and should be cleared.												
30–16	<p>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.</p> <p>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).</p> <p>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.</p> <p>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 than Maximum Packet Length.</p>												
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	<p>Multiplier Override (MultiO). 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.</p> <p>Example:</p> <p>if QH.multiplier = 3; Maximum packet size = 8; Total bytes = 15; MultiO = 0 [default]                      Three packets are sent: {Data2(8); Data1(7); Data0(0)}</p> <p>if QH.multiplier = 3; Maximum packet size = 8; Total bytes = 15; MultiO = 2                      Two packets are sent: {Data1(8); Data0(7)}</p> <p>For maximal efficiency, software should compute MultiO = greatest integer of (Total Bytes/Max. Packet Size) except for the case when Total bytes = 0; then MultiO should be 1.</p> <p><b>Note:</b> Non-ISO and non-TX endpoints must set MultiO = 00.</p>												
9–8	Reserved, should be cleared. Bits reserved for future use and should be cleared.												
7–0	<p>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:</p> <table border="1"> <thead> <tr> <th>Bit</th> <th>Status Field Description</th> </tr> </thead> <tbody> <tr> <td>7</td> <td>Active</td> </tr> <tr> <td>6</td> <td>Halted</td> </tr> <tr> <td>5</td> <td>Data Buffer Error</td> </tr> <tr> <td>3</td> <td>Transaction Error</td> </tr> <tr> <td>4,2,0</td> <td>Reserved, should be cleared</td> </tr> </tbody> </table>	Bit	Status Field Description	7	Active	6	Halted	5	Data Buffer Error	3	Transaction Error	4,2,0	Reserved, should be cleared
Bit	Status Field Description												
7	Active												
6	Halted												
5	Data Buffer Error												
3	Transaction Error												
4,2,0	Reserved, should be cleared												

Table 13-80–Table 13-82 the buffer pointer page *n* fields.

**Table 13-80. 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 13-81. 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 13-82. 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

## 13.8 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.

### 13.8.1 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.

To configure the external ULPI PHY the following initialization sequence is required.

1. After power-on reset the UTMI PHY will be in disabled state and the PLL will be held reset. The UTMI PHY should remain disabled if the ULPI is being used.
2. Set the CONTROL[PHY\_CLK\_SEL] bits to select the ULPI PHY as the source of USB controller PHY clock.
3. Wait for PHY clock to become valid. This can be determined by polling the CONTROL[PHY\_CLK\_VALID] status bit. Note that this bit is not valid once the CONTROL[USB\_EN] bit is set.

Once the PHY clock is valid the user can proceed to the device controller initialization phase.

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 to [Section 13.7, “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 DR 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 [Section 13.8.2.1, “Bus Reset.”](#)

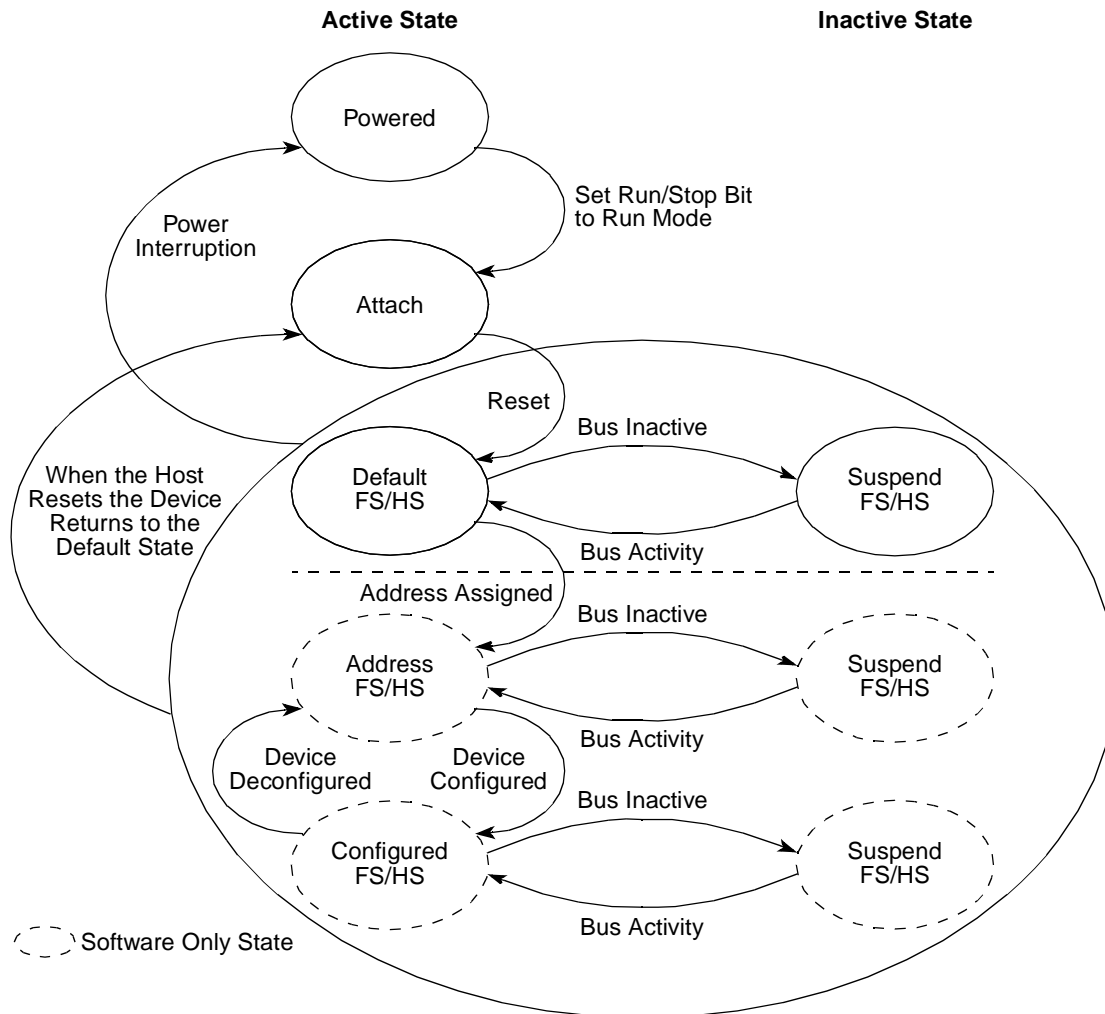
**NOTE**

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.

## 13.8.2 Port State and Control

From a chip or system reset, the USB\_DR 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*. [Figure 13-63](#) depicts the state of a USB 2.0 device.



**Figure 13-63. USB 2.0 Device States**

States powered, attach, defaultFS/HS, suspendFS/HS are implemented in the USB\_DR and are communicated to the DCD using status bits, as shown in [Table 13-83](#):

**Table 13-83. 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.

### 13.8.2.1 Bus Reset

A bus reset is used by the host to initialize downstream devices. When a bus reset is detected, the USB\_DR 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 are canceled by the device controller. The concept of priming is clarified below, but the DCD must perform the following tasks when a reset is received:

1. Clear all setup token semaphores by reading the ENDPTSETUPSTAT register and writing the same value back to the ENDPTSETUPSTAT register.
2. Clear all the endpoint complete status bits by reading the ENDPTCOMPLETE register and writing the same value back to the ENDPTCOMPLETE register.
3. Cancel all primed status by waiting until all bits in the ENDPTPRIME are 0 and then writing 0xFFFF\_FFFF to ENDPTFLUSH.
4. Read the reset bit in the PORTSC register (PORTSC[PR]) and make sure that it is still active.
  - A USB reset occurs for a minimum of 3 ms, and the DCD must reach this point in the reset cleanup before end of the reset occurs.
  - If it does not, a hardware reset of the device controller is recommended. A hardware reset can be performed by writing a one to the USB\_DR reset bit in (USBCMD[RST]). Note that 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\_DR after a hardware reset.
5. 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.

## 13.8.2.2 Suspend/Resume

This section discusses the suspend and resume functions.

### 13.8.2.2.1 Suspend Description

In order to conserve power, USB\_DR automatically enters the suspended state when no bus traffic has been observed for a specified period. When suspended, the USB\_DR 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\_DR exits suspend mode when there is bus activity. It may also request the host to exit suspend mode or selective suspend using electrical signaling to indicate remote wake-up. The ability of a device to signal remote wake-up is optional. The USB\_DR is capable of remote wake-up signaling. When the USB\_DR is reset, remote wake-up signaling must be disabled.

### 13.8.2.2.2 Suspend Operational Model

The USB\_DR 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.

### 13.8.2.2.3 Resume

If the USB\_DR is suspended, its operation is resumed when any non-idle signaling is received on its upstream facing port. In addition, the USB\_DR 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 using the Set Feature command defined in device framework (Chapter 9) of the USB 2.0 Specification.

## 13.8.3 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\_DR supports up to three 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.

### 13.8.3.1 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. Table 13-84 shows how to construct a configuration word for endpoint initialization.

**Table 13-84. Device Controller Endpoint Initialization**

Field	Value
Data Toggle Reset	1
Data Toggle Inhibit	0
Endpoint Type	00 Control 01 Isochronous 10 Bulk 11 Interrupt
Endpoint Stall	0

#### 13.8.3.1.1 Stalling

There are two occasions where the USB\_DR 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 `ENDPTCTRLn` 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 `ENDPTCTRLn` register can ensure that both stall bits are set at the same instant.

### NOTE

Any write to the `ENDPTCTRLn` register during operational mode must preserve the endpoint type field (that is, perform a read-modify-write).

Table 13-85 describes the device controller stall response matrix.

**Table 13-85. 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

## 13.8.3.2 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*.

### 13.8.3.2.1 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 `ENDPTCTRLn` register. This should only be necessary when configuring/initializing an endpoint or returning from a STALL condition.

### 13.8.3.2.2 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\_DR 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\_DR 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\_DR 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.

### 13.8.3.3 Device 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\_DR 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 expect the host will send IN requests to that endpoint. This USB\_DR 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 is used throughout the following documentation to describe the USB\_DR 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.

#### 13.8.3.3.1 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 is 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.

#### 13.8.3.3.2 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.

### 13.8.3.4 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 is 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, [Table 13-86](#), and [Table 13-87](#) 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

$$N = \text{INT}(\text{number of bytes}/\text{max. packet length}) + 1$$

With Zero Length Termination (ZLT) = 1

$$N = \text{MAXINT}(\text{number of bytes}/\text{max. packet length})$$

**Table 13-86. 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 13-87. 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 is 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 is cleared and the next pointer will be followed when the Terminate bit is clear. When the Terminate bit is set, the USB\_DR 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 is 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.

**13.8.3.4.1 Interrupt/Bulk Endpoint Bus Response Matrix**

Table 13-88 shows the interrupt/bulk endpoint bus response matrix.

**Table 13-88. Interrupt/Bulk Endpoint Bus Response Matrix**

	Stall	Not Primed	Primed	Underflow	Overflow
<b>Setup</b>	Ignore	Ignore	Ignore	N/A	N/A
<b>In</b>	STALL	NAK	Transmit	BS Error <sup>1</sup>	N/A
<b>Out</b>	STALL	NAK	Receive + NYET/ACK <sup>2</sup>	N/A	NAK
<b>Ping</b>	STALL	NAK	ACK	N/A	N/A
<b>Invalid</b>	Ignore	Ignore	Ignore	Ignore	Ignore

<sup>1</sup> Force Bit Stuff Error.

<sup>2</sup> 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.

### 13.8.3.5 Control Endpoint Operation Model

This section discusses the control endpoint operation model.

#### 13.8.3.5.1 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\_DR 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 by 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.

#### 13.8.3.5.2 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.

**13.8.3.5.3 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.

**13.8.3.5.4 Control Endpoint Bus Response Matrix**

Table 13-89 shows the device controller response to packets on a control endpoint, according to the device controller state.

**Table 13-89. Control Endpoint Bus Response Matrix**

Token Type	Endpoint State					Setup Lockout
	Stall	Not Primed	Primed	Underflow	Overflow	
Setup	ACK	ACK	ACK	N/A	SYSERR <sup>1</sup>	
In	STALL	NAK	Transmit	BS Error <sup>2</sup>	N/A	N/A
Out	STALL	NAK	Receive + NYET/ACK <sup>3</sup>	N/A	NAK	N/A

**Table 13-89. 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

<sup>1</sup> SYSERR—System error should never occur when the latency FIFOs are correctly sized and the DCD is responsive.

<sup>2</sup> Force Bit Stuff Error.

<sup>3</sup> NYET/ACK—NYET unless the Transfer Descriptor has packets remaining according to the USB variable length protocol then ACK.

### 13.8.3.6 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\_DR will be accomplished by the following:

- Exactly MULT Packets per (micro)Frame are transmitted/received. Note that 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 are sent in response to an IN request to an unprimed endpoint. 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 is held ready until executed or canceled by the DCD.

The USB\_DR 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 is cleared as usual to indicate to software that the device controller completed 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.

**13.8.3.6.1 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\_DR 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.

#### 13.8.3.6.2 Isochronous Endpoint Bus Response Matrix

Table 13-90 shows the isochronous endpoint bus response matrix.

Table 13-90. Isochronous Endpoint Bus Response Matrix

	Stall	Not Primed	Primed	Underflow	Overflow
Setup	STALL	STALL	STALL	N/A	N/A
In	NULL <sup>1</sup> Packet	NULL Packet	Transmit	BS Error <sup>2</sup>	N/A
Out	Ignore	Ignore	Receive	N/A	Drop Packet
Ping	Ignore	Ignore	Ignore	Ignore	Ignore
Invalid	Ignore	Ignore	Ignore	Ignore	Ignore

<sup>1</sup> Zero Length Packet.

<sup>2</sup> Force Bit Stuff Error.

#### 13.8.4 Managing Queue Heads

Figure 13-64 shows the endpoint queue head diagram.

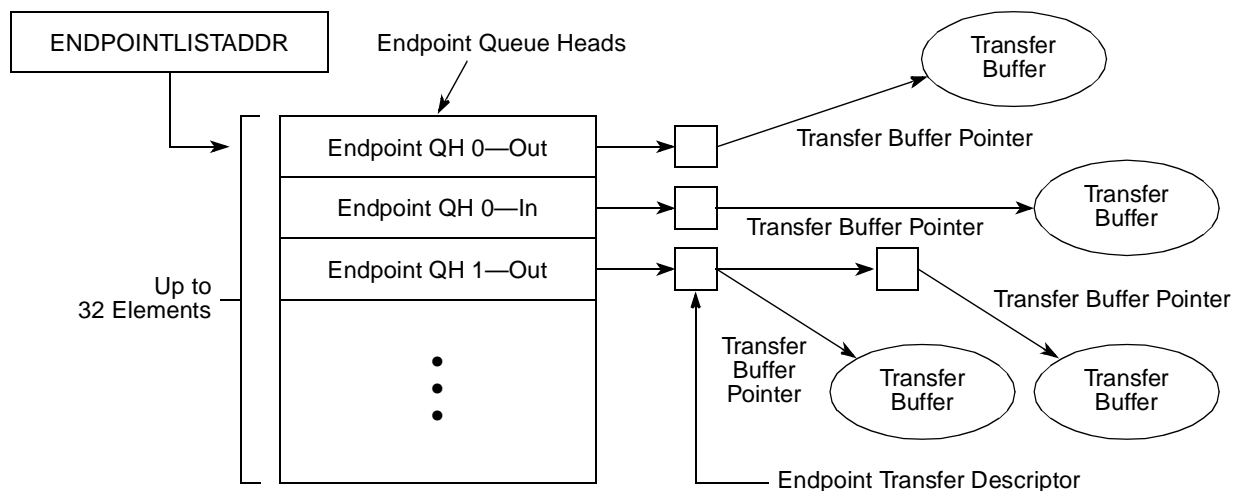


Figure 13-64. Endpoint Queue Head Diagram

The device queue head (dQH) points to the linked list of transfer tasks, each depicted by the device Transfer Descriptor (dTDD). An area of memory pointed to by ENDPOINTLISTADDR contains a group of all dQHs in a sequential list as shown in [Figure 13-64](#). 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 13.8.5.1, "Software Link Pointers"](#)).

In addition to the current and next pointers and the dTD overlay, the dQH also contains the following parameters for the associated endpoint: Multiplier, Maximum Packet Length, Interrupt On Setup. The complete initialization of the dQH including these fields is demonstrated in the next section.

### 13.8.4.1 Queue Head Initialization

One pair of device queue heads must be initialized for each active endpoint. To initialize a device queue head:

- Write the MaxPacketSize 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 and in conjunction with the USB Chapter 9 protocol. Note that 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.

### 13.8.4.2 Operational Model for Setup Transfers

As discussed in [Section 13.8.3.5, "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 13.8.5.5, "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.

## 13.8.5 Managing Transfers with Transfer Descriptors

### 13.8.5.1 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 13-65 shows the software link pointers.

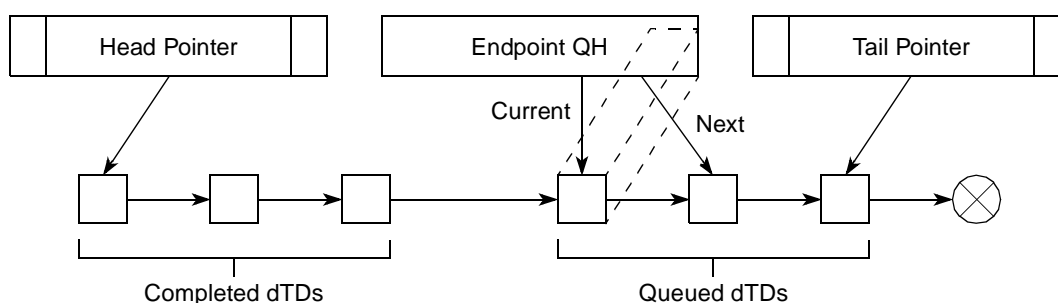


Figure 13-65. Software Link Pointers

### 13.8.5.2 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 seven 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.

### 13.8.5.3 Executing a Transfer Descriptor

To safely add a dTD, the DCD must account for the event in which 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.

First, determine whether the link list is empty by checking the DCD driver to see if the pipe is empty (internal representation of linked-list should indicate if any packets are outstanding). Then follow the sequence of actions in the following list as appropriate, depending on whether the link list is empty or not empty.

- Link list is empty
  - a) Write dQH next pointer AND dQH terminate bit to ‘0’ as a single DWord operation.
  - b) Clear active and halt bit in dQH (in case set from a previous error).
  - c) Prime endpoint by writing ‘1’ to correct bit position in ENDPTPRIME.
- Link list is not empty
  - a) Add dTD to end of linked list.
  - b) Read correct prime bit in ENDPTPRIME—if ‘1’ DONE.
  - c) Set ATDTW bit in USBCMD register to ‘1’.
  - d) Read correct status bit in ENDPTSTATUS. (store in tmp. variable for later).
  - e) Read ATDTW bit in USBCMD register.
    - If ‘0’ goto 3.
    - If ‘1’ continue to 6.
  - f) Write ATDTW bit in USBCMD register to ‘0’.
  - g) If status bit read in (3) is ‘1’ DONE.
  - h) If status bit read in (3) is ‘0’ then Goto Case 1: Step 1.

### 13.8.5.4 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 is 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.

#### 13.8.5.5 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 by repeating steps 1–3 until each endpoint is successfully flushed.

### 13.8.5.6 Device Error Matrix

Table 13-91 summarizes packet errors that are not automatically handled by the USB controller.

**Table 13-91. 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 13-92 provides the error descriptions.

**Table 13-92. Error Descriptions**

<b>Overflow</b>	Number of bytes received exceeded max. packet size or total buffer length. <b>Note:</b> This error also sets the Halt bit in the dQH. If there are dTDs remaining in the linked list for the endpoint, they will not be executed.
<b>ISO Packet Error</b>	CRC Error on received ISO packet. Contents not guaranteed to be correct.
<b>ISO Fulfillment Error</b>	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.

## 13.8.6 Servicing Interrupts

The interrupt service routine must consider that there are high-frequency, low-frequency operations, and error operations and order accordingly.

### 13.8.6.1 High-Frequency Interrupts

High frequency interrupts in particular should be handed in the order shown in Table 13-93. The most important of these is listed first because the DCD must acknowledge a setup buffer in the timeliest manner possible.

**Table 13-93. Interrupt Handling Order**

Execution Order	Interrupt	Action
1a	USB Interrupt <sup>1</sup> ENDPTSETUPSTATUS	Copy contents of setup buffer and acknowledge setup packet (as indicated in Section 13.8.4, "Managing Queue Heads"). Process setup packet according to USB 2.0 Chapter 9 or application specific protocol.

**Table 13-93. Interrupt Handling Order (continued)**

Execution Order	Interrupt	Action
1b	USB Interrupt ENDPTCOMPLETE	Handle completion of dTD as indicated in <a href="#">Section 13.8.4, “Managing Queue Heads”</a> .
2	SOF Interrupt	Action as deemed necessary by application. This interrupt may not have a use in all applications.

<sup>1</sup> It is likely that multiple interrupts to stack up on any call to the Interrupt Service Routine AND during the Interrupt Service Routine.

### 13.8.6.2 Low-Frequency Interrupts

The low frequency events include the interrupts shown in [Table 13-94](#). These interrupts can be handled in any order because they do not occur often in comparison to the high-frequency interrupts.

**Table 13-94. 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.

### 13.8.6.3 Error Interrupts

Error interrupts are the least frequently occurring events. They should be placed last in the interrupt service routine. [Table 13-95](#) shows the error interrupt events.

**Table 13-95. 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.

## 13.9 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.
- For the purposes of the DR implementing dual-role host/device controller with support for OTG applications, it is necessary to deviate from the EHCI specification. Device operation and OTG operation are not specified in the EHCI and thus the implementation supported in the DR module is proprietary.

### 13.9.1 Embedded Transaction Translator Function

The DR 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.

#### 13.9.1.1 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 13.3.1.3, “Host Controller Structural Parameters \(HSCPARAMS\),”](#) for usage information.

#### 13.9.1.2 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.

#### 13.9.1.3 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 always sets the port enable after the port reset operation regardless of the result of the host device chirp result. The resulting port speed is 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. Table 13-96 summarizes the functional differences between EHCI and EHCI with embedded TT.

**Table 13-96. 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)]

### 13.9.1.4 Data Structures

The same data structures used for FS/LS transactions through a HS hub are also used for transactions through the Root Hub. The following list demonstrates how the Hub Address and Endpoint Speed fields should be set for directly attached FS/LS devices and hubs:

- 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 is sent before the transmitting LS traffic.

Maximum Packet Size must be less than or equal 64 or undefined behavior may result.

- 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.

### 13.9.1.5 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 DR 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.

### 13.9.1.5.1 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 are 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).

### 13.9.1.5.2 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 13-97](#) summarizes the conditions where handshakes are emulated from internal state instead of actual handshakes to HS split bus traffic.

**Table 13-97. Emulated Handshakes**

Condition	Emulate TT Response
<b>Start-Split:</b> All asynchronous buffers full	NAK
<b>Start-Split:</b> All periodic buffers full	ERR
<b>Start-Split:</b> Success for start of Async. Transaction	ACK
<b>Start-Split:</b> Start Periodic Transaction	No Handshake (Ok)
<b>Complete-Split:</b> Failed to find transaction in queue	Bus Time Out
<b>Complete-Split:</b> Transaction in Queue is Busy	NYET
<b>Complete-Split:</b> Transaction in Queue is Complete	[Actual Handshake from FS/LS device]

### 13.9.1.5.3 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

### 13.9.1.5.4 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.

### 13.9.1.5.5 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 13.3.1.3, “Host Controller Structural Parameters \(HCSPARAMS\),”](#) for more information.

## 13.9.2 Device Operation

The co-existence of a device operational controller within the DR module has little effect on EHCI compatibility for host operation except as noted in this section.

### 13.9.3 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 DR 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 DR module registers).

### 13.9.4 SOF Interrupt

The SOF interrupt is a free running 125  $\mu$ sec 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 13.3.2.2, “USB Status Register \(USBSTS\),”](#) and [Section 13.3.2.3, “USB Interrupt Enable Register \(USBINTR\),”](#) for more information.

### 13.9.5 Embedded Design

This is an Embedded USB Host Controller as defined by the EHCI specification and thus does not implement the PCI configuration registers.

#### 13.9.5.1 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  $\mu$ sec 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.

### 13.9.6 Miscellaneous Variations from EHCI

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.

#### 13.9.6.1 Discovery

This section discusses port reset and port speed detection.

##### 13.9.6.1.1 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 is now operational and at this point the port speed has been determined.

#### 13.9.6.1.2 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

### 13.10 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 13-66 shows ULPI timing.

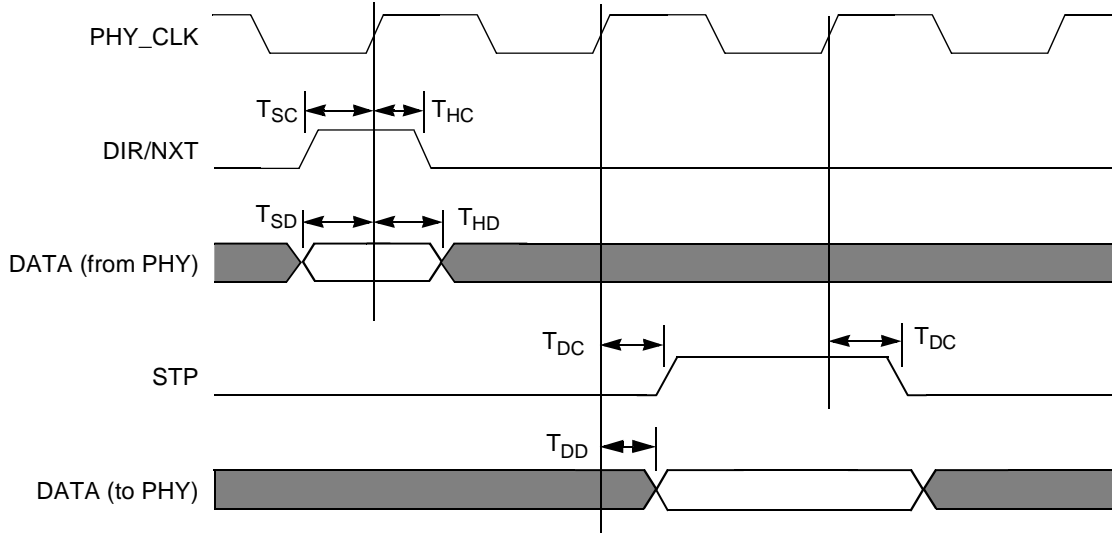


Figure 13-66. ULPI Timing

Table 13-98 summarizes the ULPI timing parameters.

Table 13-98. ULPI Timing

Parameter	Symbol	Min	Max	Units
Control signal setup time	T <sub>SC</sub>	—	4	ns
Data setup time	T <sub>SD</sub>	—	4	ns
Control signal hold time	T <sub>HC</sub>	0	—	ns
Data hold time	T <sub>HD</sub>	0	—	ns
Control output delay	T <sub>DC</sub>	2	7	ns
Data output delay	T <sub>DD</sub>	2	7	ns

Figure 13-67 shows the diagram for the sending of RX CMD.

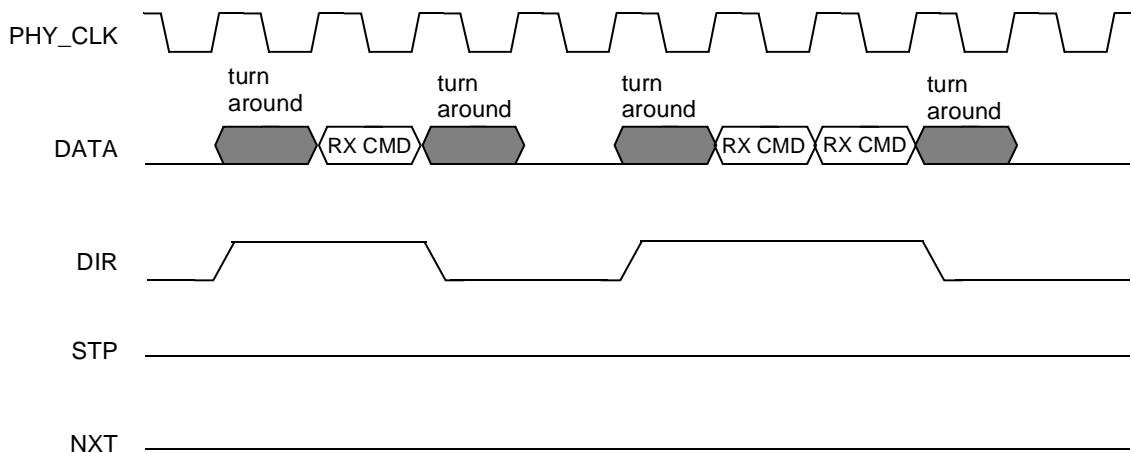


Figure 13-67. Sending of RX CMD

Figure 13-68 shows ULPI data transmit—NOPID.

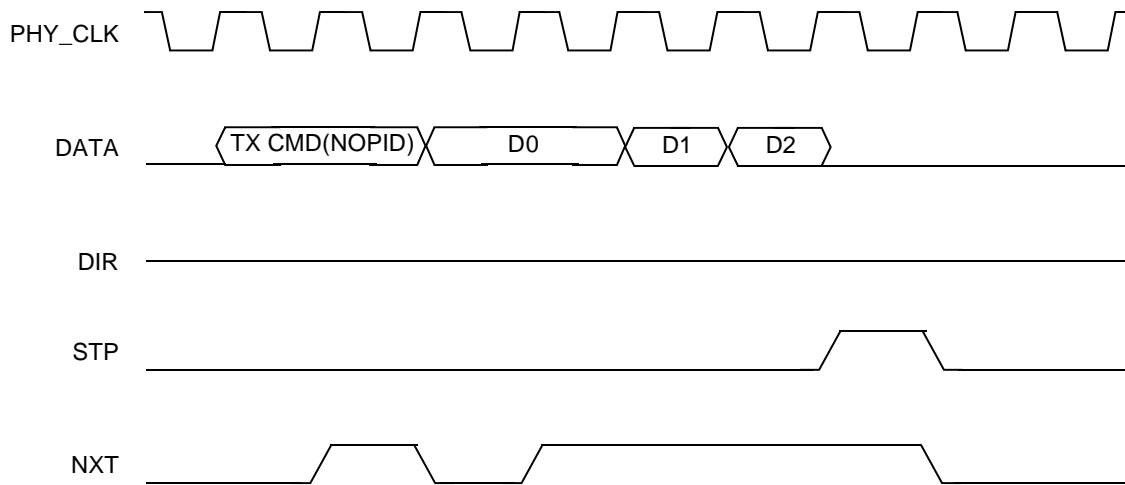


Figure 13-68. ULPI Data Transmit (NOPID)

Figure 13-69 shows ULPI data transmit—PID.

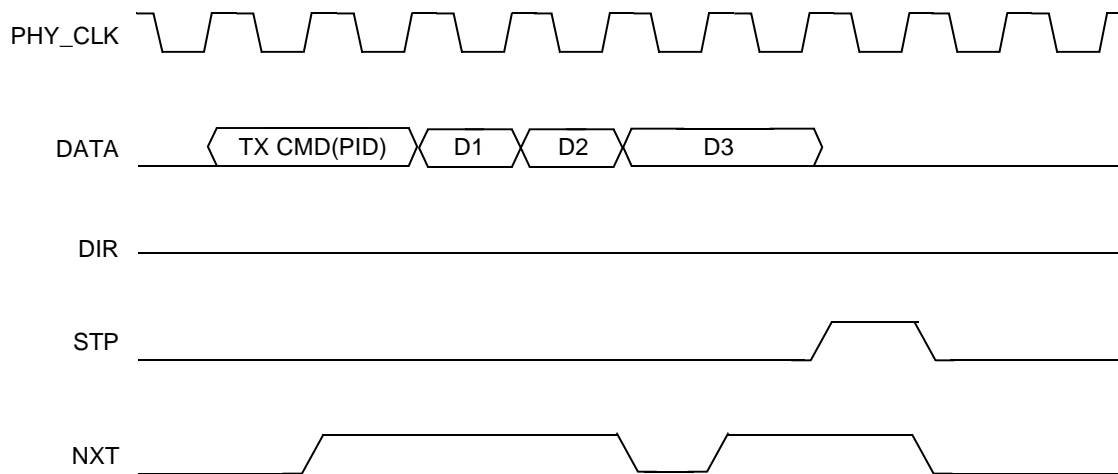


Figure 13-69. ULPI Data Transmit (PID)

Figure 13-70 shows ULPI data receive.

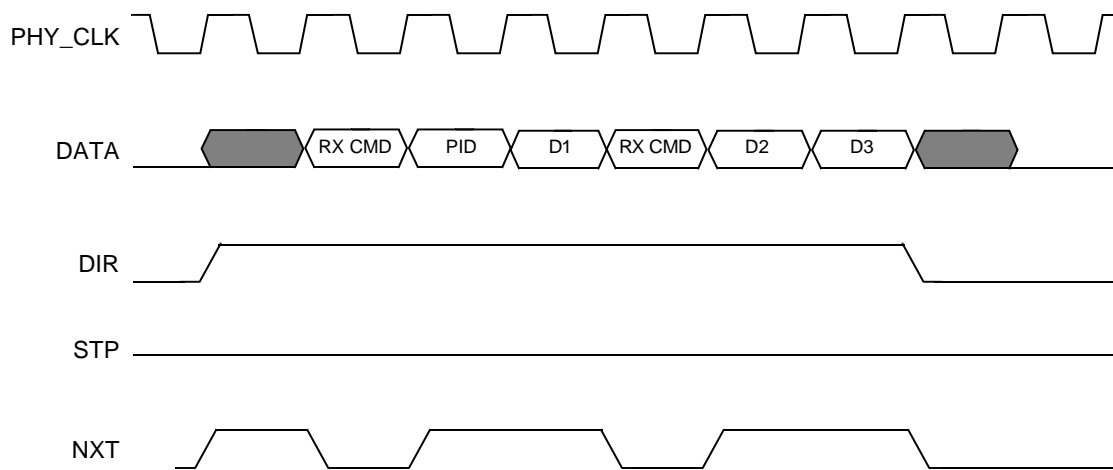


Figure 13-70. ULPI Data Receive



Figure 13-71 shows ULPI register write.

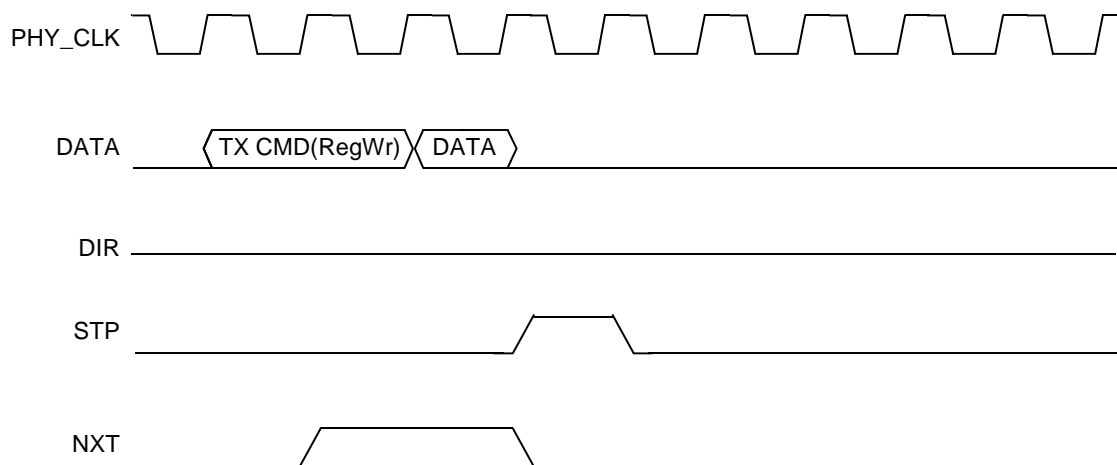


Figure 13-71. ULPI Register Write

Figure 13-72 shows ULPI register read.

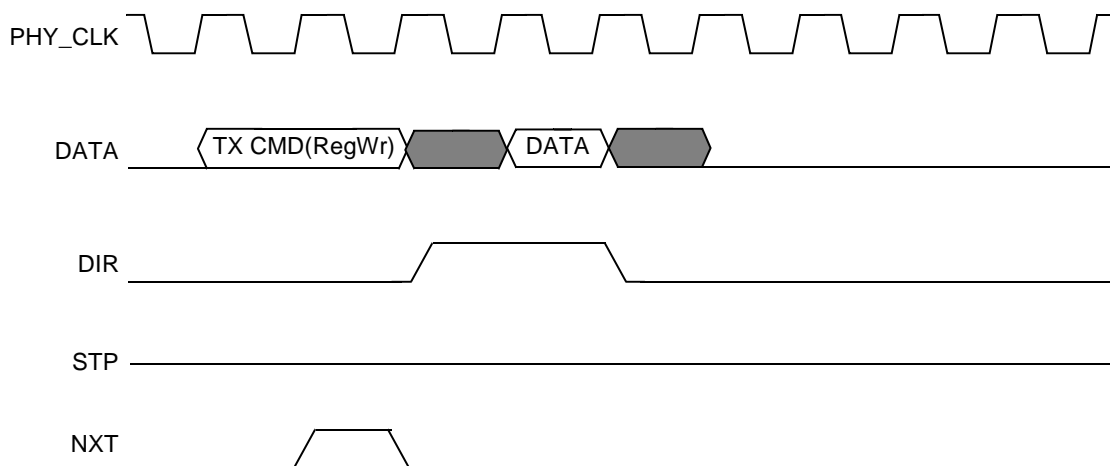


Figure 13-72. ULPI Register Read



# Chapter 14

## PCI Express Interface Controller

The MPC8308 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

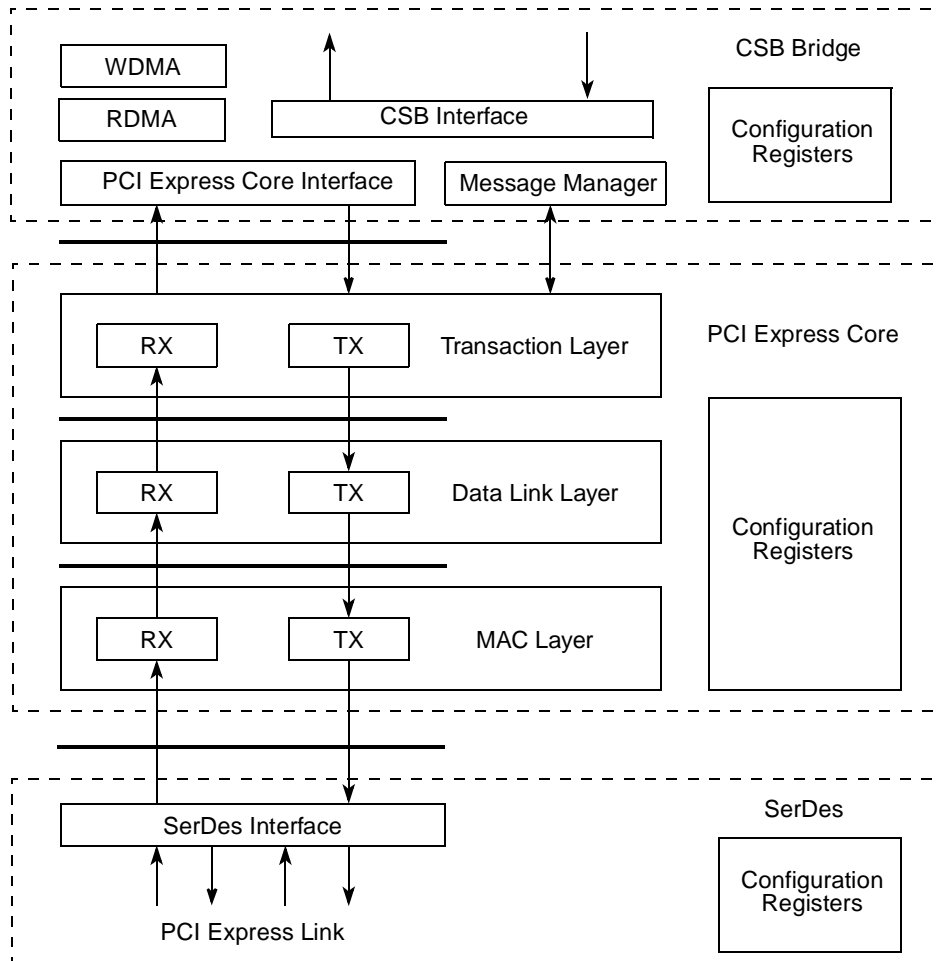
- Much of the available PCI Express literature refers to a 16-bit quantity as a WORD and a 32-bit quantity as a DWORD. This is inconsistent with the terminology in the rest of this manual where the terms ‘word’ and ‘double word’ refer to a 32-bit and 64-bit quantity, respectively. Where necessary to avoid confusion, the precise number of bits or bytes is specified.
- The PCI Express engine does not support misaligned byte transfers. It must be DWORD aligned to the CSB bus.

### 14.1 Introduction

The PCI Express controller is a mechanism for communicating with PCI Express devices. The controller contains three major parts:

- PCI Express core—Handles the transaction, data link and MAC layers and contains the configuration header and control registers.
- CSB bridge—Controls the transfer of the transactions between the PCI Express transaction layer and the CSB, and include Write and Read DMA engines, a message manager and a set of configuration registers.
- SerDes—Controls the transfer between the PCI Express MAC layer and the physical link, and includes another set of configuration registers (described in [Chapter 15, “SerDes PHY”](#)).

Figure 14-1 is a high-level block diagram of the PCI Express controller.



**Figure 14-1. PCI Express Controller Block Diagram**

The PCI Express controller connects the coherent system bus (CSB) to the PCI Express bus, which is a 2.5-GHz serial interface that supports  $\times 1$  lane. As both a master (initiator) and a target device, the PCI Express interface is capable of high bandwidth data transfer and is designed to support the next generation of I/O devices. When it comes out of reset, the PCI Express interface performs link width negotiation and exchanges flow control credits with its link partner. When link auto-negotiation finishes, the controller is ready for operation.

Internally, the design contains queues to keep track of inbound and outbound transactions. The control logic handles buffer management, bus protocol, transaction spawning, and tag generation. In addition, memory blocks store inbound and outbound data.

The device can be configured to operate in either root complex (RC) or endpoint (EP) mode. An RC device connects the CPU/memory subsystem to the I/O devices, while an EP device typically denotes a peripheral or I/O device. In RC mode, a type 1 configuration header is used. In EP mode, a type 0 configuration header is used.

As an initiator, the device supports memory read and write operations with a maximum payload of 128 bytes and I/O transactions. In addition, outbound configurations are supported if the device is in RC mode. As a target interface, the device accepts read and write operations to local memory space. Furthermore, as an EP device, the device accepts configuration transactions to the internal PCI Express configuration registers. Inbound I/O transactions are not supported.

### 14.1.1 MPC8308 as a PCI Express Initiator

Outbound CSB transactions to PCI Express are first mapped to a translation window to determine which PCI Express transactions are to be issued. A transaction from the CSB can become a memory, I/O, or configuration transaction on the PCI Express bus depending on the window attributes. A transaction can be broken up into smaller transactions depending on the original request size, transaction type, PCI Express device control register's Max\_Payload\_Size field (for writes) and PCI Express device control register's Max\_Read\_Request\_Size field (for reads). The device performs PCI Express ordering rule checks to determine the next transaction to be sent on the PCI Express bus. In general, transactions are serviced in the order they are received from the CSB. The device allows reordering of higher-priority transactions to bypass lower-priority transactions only when there is a stall condition. For posted write transactions, after all data is received on the CSB, the data is forwarded to the PCI Express bus and the transaction is considered to be complete. For non-posted write transactions, the device waits for a completion to return from the link partner before considering the transaction to be complete. For non-posted read transactions, the device waits for all completion packets to return from the link partner and then forwards all data back to the CSB before terminating the transaction.

There are two methods of generating PCI Express outbound transactions:

- One of the CSB masters, such as the e300 host, directly initiates a transaction. This is referred to as "PIO."
- The write or read DMA engines, which are part of the PCI Express controller CSB bridge, is used.

The DMA method is more efficient for transferring large chunks of data. The outbound windows are used and shared by both methods.

### 14.1.2 MPC8308 as a PCI Express Target

Inbound PCI Express transactions to the CSB are first mapped to the CSB address space through a translation window. A transaction can be broken up into smaller transactions when it is sent to the CSB depending on the original size, byte enables, and starting/ending addresses. The device performs PCI Express ordering rule checks to determine the next transaction to be sent to the CSB. In general, transactions are serviced in the order they are received from the PCI Express bus. The device allows reordering of higher-priority transactions to bypass lower-priority transactions only when there is a stall condition. For posted write transactions, after all data is received on the PCI Express bus, the data is forwarded to the CSB and the transaction is considered to be complete. For non-posted read transactions, the device waits for enough completion packets (dependent on the packet length) to return and then forwards data back to the PCI Express bus. This process continues until there are no more completion packets left to be sent.

### 14.1.3 Features

The following is a list of PCI Express controller features:

- Designed to be compatible with the *PCI Express Base Specification, Version 1.0a*
- Root complex (RC) and endpoint (EP) configurations
- 32- and 64-bit address support
- PCI Express link of  $\times 1$  lane
- Access to all PCI Express memory
- Access to I/O address spaces as requestor only in RC mode
- Posting of processor-to-PCI Express and PCI Express-to-memory writes
- Strong and relaxed transaction ordering rules
- PCI Express configuration registers
- Baseline and advanced error reporting
- One virtual channel (VC0)
- 128-byte maximum payload size (Max\_Payload\_Size) for memory read and write operations
- Four inbound general-purpose translation windows
- Four outbound translation windows
- Up to four outstanding PCI Express transactions from each controller (posted or non-posted)
- Credit-based flow control management
- PCI Express messages and interrupts
- Maximum 32-byte payload transactions from the CSB
- Interrupt generation from messages or upon detection of errors
- Read and Write DMA engines
- Support polarity inversion

### 14.1.4 Modes of Operation

This section describes how some parameters that affect the PCI Express controller operating modes are determined by dedicated memory mapped registers.

#### 14.1.4.1 Root Complex/Endpoint Modes

The PCI Express controller can function as either a root complex (RC) or an endpoint (EP) device. The PCI Express control registers 1 and 2 determine the RC/EP mode; see [Section 5.2.2.11, “PCI Express Control Registers \(PECR1\).”](#)

#### 14.1.4.2 Link Width

The link width of the PCI Express controller is  $\times 1$ .

### 14.1.4.3 Reference Clock

The reference clock for the PCI Express PHY is set to 100 MHz upon POR. To change the reference clock frequency after POR, program the SRDSCR4 (see [Section 15.3.5, “SerDes Control Register 4 \(SRDSCR4\)”](#)) and initiate a SerDes PHY reset sequence.

By default, PCI Express controller clock is the same as CSB clock.

## 14.2 External Signal Descriptions

PCI Express defines the connection between two devices as a link, which can be composed of a single lane or multiple lanes. Each lane consists of a differential pair for transmitting ( $TX_n$  and  $\overline{TX}_n$ ) and a differential pair for receiving ( $RX_n$  and  $\overline{RX}_n$ ) with an embedded data clock.

[Table 14-1](#) describes the external PCI Express interface signals.

**Table 14-1. PCI Express Interface Signals—Detailed Signal Descriptions**

Signal	I/O	Description
$RXA/\overline{RXA}$	I	Receive data. Receive data differential signal pair carry PCI Express packet information.
$TXA/\overline{TXA}$	O	Transmit data. The transmit data differential signal pair carry PCI Express packet information.

## 14.3 Memory Map/Register Definitions

The PCI Express interface supports the following register types:

- Memory-mapped registers—Control PCI Express address translation, PCI Express error management, and PCI Express configuration register access on the device. These registers are described in [Section 14.3.1, “PCI Express Memory Map.”](#)
- PCI Express configuration registers within the PCI Express configuration header—Specified by the PCI Express specification for every PCI Express device. They are described in [Section 14.4.1, “Common PCI Express-Compatible Configuration Header Registers.”](#)

From the PCI Express side, the configuration header registers can be accessed through configuration access, and the memory-mapped registers can be accessed through memory transactions after the inbound translation window is programmed. From the CSB side, all these registers are memory-mapped.

### 14.3.1 PCI Express Memory Map

The PCI Express memory-mapped registers, listed in [Table 14-3](#), are accessed by reading and writing to an address composed of the base address (specified in the IMMRBAR on the CSB side or the ATMU windows on the PCI Express side) plus the offset of the specific register to be accessed. In this table and in the register figures and fields description, 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.

Memory-mapped registers for PCI Express controller begin at block base address 0x0\_9000.

Table 14-2 below lists the address ranges for each type of register. Undefined address spaces are reserved.

**Table 14-2. PCI Express Controller Register Groups**

Register	Offset Range	Section/Page
<b>PCI Express Core Registers</b>		
Common PCI-Compatible Configuration Header Registers	0x000–0x3FF	<a href="#">14.4.1/14-14</a>
PCI Express Core Control and Status Registers (CSRs)	0x400–0x4CF	<a href="#">14.4.6/14-62</a>
PCI Express BAR Configuration Registers (EP Mode)	0x4D8–0x4FF	<a href="#">14.4.7/14-72</a>
PCI Express Extended Status and Control Registers	0x590–0x7FF	<a href="#">14.4.8/14-74</a>
<b>PCI Express CSB Bridge Registers</b>		
Global Registers	0x800–0x83F	<a href="#">14.5.2/14-77</a>
PCI Express Outbound PIO Registers	0x840–0x8DF	<a href="#">14.5.3/14-79</a>
PCI Express Inbound PIO Registers	0x8E0–0x87F	<a href="#">14.5.4/14-81</a>
PCI Express DMA Registers	0x990–0xADF	<a href="#">14.5.5/14-83</a>
Mailbox Registers	0xB20–0xB9F	<a href="#">14.5.6/14-87</a>
PCI Express Host Interrupt Registers	0xBA0–0xBDF	<a href="#">14.5.7/14-89</a>
CSB System Interrupt Registers	0xBE0–0xC1F	<a href="#">14.5.8/14-94</a>
PCI Express Outbound Address Mapping Registers	0xCA0–0xDDF	<a href="#">14.5.10/14-103</a>
PCI Express EP Inbound Address Translation Registers	0xDE0–0xE5F	<a href="#">14.5.11/14-106</a>
PCI Express RC Inbound Address Mapping Registers	0xE60–0xFFF	<a href="#">14.5.12/14-107</a>

**Table 14-3. PCI Express Memory Map**

Offset	Register	Access	Reset	Section/Page
<b>PCI Express—Block Base Address 0x0_9000</b>				
<b>PCI Express Controller Registers</b>				
<b>PCI Express Core Configuration Header Registers</b>				
0x000	PCI Express Vendor ID Register	R	0x1957	<a href="#">14.4.1.1/14-15</a>
0x002	PCI Express Device ID Register	R	Device-specific	<a href="#">14.4.1.2/14-15</a>
0x004	PCI Express Command Register	Mixed	0x0000	<a href="#">14.4.1.3/14-15</a>
0x006	PCI Express Status Register	Mixed	0x0010	<a href="#">14.4.1.4/14-17</a>
0x008	PCI Express Revision ID Register	R	Revision-specific	<a href="#">14.4.1.5/14-18</a>
0x009	PCI Express Class Code Register	Mixed	0x0B20	<a href="#">14.4.1.6/14-18</a>



Table 14-3. PCI Express Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
<b>PCI Express—Block Base Address 0x0_9000</b>				
0x00C	PCI Express Cache Line Size Register	R/W	0x00	14.4.1.7/14-19
0x00D	PCI Express Latency Timer Register	R	0x00	14.4.1.8/14-19
0x00E	PCI Express Header Type Register	R	0x00 (EP mode) 0x01 (RC mode)	14.4.1.9/14-20
0x00F	PCI Express BIST Register	R	0x00	14.4.1.10/14-21
0x010– 0x014	Base Address Registers 0 and 1 (BAR0/BAR1) (EP mode only)	Mixed	0x0008	14.4.2.1.1/14-22
0x018– 0x020	Base Address Registers 2 and 4 (BAR2/BAR4) (EP mode only)	Mixed	0x0000_000C	14.4.2.1.2/14-23
0x01C– 0x024	Base Address Registers 3 and 5 (BAR3/BAR5) (EP mode only)	R/W	0x0000_0000	14.4.2.1.3/14-23
0x02C	PCI Express Subsystem Vendor ID Register (EP mode only)	Special	0x0000	14.4.2.2/14-24
0x02E	PCI Express Subsystem ID Register (EP mode only)	Special	0x0000	14.4.2.3/14-24
0x034	PCI Express Capabilities Pointer Register	R	0x0044	14.4.2.4/14-25
0x03C	PCI Express Interrupt Line Register (EP mode only)	R/W	0x0000	14.4.2.5/14-25
0x03D	Reserved	—	—	—
0x03E	PCI Express Minimum Grant Register (EP mode only)	R	0x0000	14.4.2.6/14-26
0x03F	PCI Express Maximum Latency Register (EP mode only)	R	0x0000	14.4.2.7/14-26
0x018	PCI Express Primary Bus Number Register (RC mode only)	R/W	0x0000	14.4.3.1/14-27
0x019	PCI Express Secondary Bus Number Register (RC mode only)	R/W	0x0000	14.4.3.2/14-28
0x01A	PCI Express Subordinate Bus Number Register (RC mode only)	R/W	0x0000	14.4.3.3/14-28
0x01B	Secondary Latency Timer Register 2 (RC mode only)	R	0x0000	14.4.3.4/14-28
0x01C	PCI Express I/O Base Register (RC mode only)	R	0x0000	14.4.3.5/14-29
0x01D	PCI Express I/O Limit Register (RC mode only)	R	0x0000	14.4.3.6/14-29
0x01E	PCI Express Secondary Status Register (RC mode only)	Mixed	0x0000	14.4.3.7/14-30
0x020	<i>PCI Express Memory Base Register (RC mode only)<sup>1</sup></i>	R/W	0x0000	14.4.3.8/14-30
0x022	<i>PCI Express Memory Limit Register (RC mode only)<sup>1</sup></i>	R/W	0x0000	14.4.3.9/14-31
0x024	<i>PCI Express Prefetchable Memory Base Register (RC mode only)<sup>1</sup></i>	R/W	0x0000	14.4.3.10/14-31
0x026	<i>PCI Express Prefetchable Memory Limit Register (RC mode only)<sup>1</sup></i>	R/W	0x0000	14.4.3.11/14-32
0x028	<i>PCI Express Prefetchable Base Upper 32-Bit Register (RC mode only)<sup>1</sup></i>	R/W	0x0000	14.4.3.12/14-32
0x02C	<i>PCI Express Prefetchable Limit Upper 32-Bit Register (RC mode only)<sup>1</sup></i>	R/W	0x0000	14.4.3.13/14-33
0x030	<i>PCI Express I/O Base Upper 16-Bit Register (RC mode only)<sup>1</sup></i>	R	0x0000	14.4.3.14/14-33
0x032	<i>PCI Express I/O Limit Upper 16-Bit Register (RC mode only)<sup>1</sup></i>	R	0x0000	14.4.3.15/14-34

Table 14-3. PCI Express Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
<b>PCI Express—Block Base Address 0x0_9000</b>				
0x034	PCI Express Capabilities Pointer Register	R	0x044	<a href="#">14.4.3.16/14-34</a>
0x03C	PCI Express Interrupt Line Register	R/W	0x0000	<a href="#">14.4.3.17/14-35</a>
0x03D	Reserved	—	—	—
0x03E	PCI Express Bridge Control Register (RC mode only)	R/W	0x0000	<a href="#">14.4.3.18/14-35</a>
0x044	PCI Express Power Management Capability ID Register	R	0x01	<a href="#">14.4.4.1/14-37</a>
0x045	PCI Express Power Management Next Capabilities Pointer Register	R	0x4C	<a href="#">14.4.4.2/14-37</a>
0x046	PCI Express Power Management Capabilities Register	R	0x7E02	<a href="#">14.4.4.3/14-38</a>
0x048	PCI Express Power Management Status and Control Register	Mixed	0x0000	<a href="#">14.4.4.4/14-38</a>
0x04B	PCI Express Power Management Data Register	R	0x0000	<a href="#">14.4.4.5/14-39</a>
0x04C	PCI Express Capability ID Register	R	0x10	<a href="#">14.4.4.6/14-39</a>
0x04D	PCI Express Next Capabilities Pointer Register	R	0x70	<a href="#">14.4.4.7/14-40</a>
0x04E	PCI Express Capabilities Register	R	0x00n1	<a href="#">14.4.4.8/14-40</a>
0x050	PCI Express Device Capabilities Register	R	0x0000_0000	<a href="#">14.4.4.9/14-41</a>
0x054	PCI Express Device Control Register	R/W	0x2810	<a href="#">14.4.4.10/14-42</a>
0x056	PCI Express Device Status Register	Mixed	0x0000	<a href="#">14.4.4.11/14-43</a>
0x058	PCI Express Link Capabilities Register	R	0x0003_D411	<a href="#">14.4.4.12/14-43</a>
0x05C	PCI Express Link Control Register	R/W	0x0000	<a href="#">14.4.4.13/14-44</a>
0x05E	PCI Express Link Status Register	R	0x0011	<a href="#">14.4.4.14/14-45</a>
0x060	PCI Express Slot Capabilities Register	R	0x000007c0	<a href="#">14.4.4.15/14-45</a>
0x064	PCI Express Slot Control Register	R/W	0x0000	<a href="#">14.4.4.16/14-46</a>
0x066	PCI Express Slot Status Register	Mixed	0x0040	<a href="#">14.4.4.17/14-47</a>
0x068	PCI Express Root Control Register (RC mode only)	R/W	0x0000	<a href="#">14.4.4.18/14-47</a>
0x06C	PCI Express Root Status Register (RC mode only)	Mixed	0x0000_0000	<a href="#">14.4.4.19/14-48</a>
0x070	PCI Express MSI Message Capability ID Register (EP mode only)	R	0x05	<a href="#">14.4.4.20/14-49</a>
0x072	PCI Express MSI Message Control Register (EP mode only)	Mixed	0x0088	<a href="#">14.4.4.21/14-49</a>
0x074	PCI Express MSI Message Address Register (EP mode only)	R/W	0x0000_0000	<a href="#">14.4.4.22/14-50</a>
0x078	PCI Express MSI Message Upper Address Register (EP mode only)	R/W	0x0000_0000	<a href="#">14.4.4.23/14-50</a>
0x07C	PCI Express MSI Message Data Register (EP mode only)	R/W	0x0000	<a href="#">14.4.4.24/14-50</a>
0x100	PCI Express Advanced Error Reporting Capability ID Register	R	0x1381_0001	<a href="#">14.4.5.1/14-53</a>
0x104	PCI Express Uncorrectable Error Status Register	R/W	0x0000_0000	<a href="#">14.4.5.2/14-53</a>
0x108	PCI Express Uncorrectable Error Mask Register	R/W	0x0000_0000	<a href="#">14.4.5.3/14-54</a>
0x10C	PCI Express Uncorrectable Error Severity Register	R/W	0x0006_2010	<a href="#">14.4.5.4/14-55</a>

Table 14-3. PCI Express Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
<b>PCI Express—Block Base Address 0x0_9000</b>				
0x110	PCI Express Correctable Error Status Register	w1c	0x0000_0000	<a href="#">14.4.5.5/14-56</a>
0x114	PCI Express Correctable Error Mask Register	R/W	0x0000_0000	<a href="#">14.4.5.6/14-57</a>
0x118	PCI Express Advanced Error Capabilities and Control Register	R/W	0x0000_00A0	<a href="#">14.4.5.7/14-58</a>
0x11C	PCI Express Header Log Register	R	0x0000_0000	<a href="#">14.4.5.8/14-58</a>
0x120	PCI Express Header Log Register	R	0x0000_0000	
0x124	PCI Express Header Log Register	R	0x0000_0000	
0x128	PCI Express Header Log Register	R	0x0000_0000	
0x12C	PCI Express Root Error Command Register	R/W	0x0000_0000	<a href="#">14.4.5.9/14-60</a>
0x130	PCI Express Root Error Status Register	Mixed	0x0000_0000	<a href="#">14.4.5.10/14-60</a>
0x134	PCI Express Error Source Identification Register	R	0x0000_0000	<a href="#">14.4.5.11/14-61</a>
<b>PCI Express Core Control and Status Registers (CSRs)</b>				
0x404	PCI Express LTSSM State Status Register (PEX_LTSSM_STAT)	R	0x0000_0000	<a href="#">14.4.6.1/14-62</a>
0x41C	PCI Express N_FTS Control Register (PEX_NFTS_CTRL)	R/W	0x0000_4040	<a href="#">14.4.6.2/14-63</a>
0x438	PCI Express ACK Replay Timeout Register (PEX_ACKRPLY_TO)	R/W	0x003B_2090	<a href="#">14.4.6.3/14-64</a>
0x440	PCI Express Core Clock Ratio Register (PEX_GCLK_RATIO)	Mixed	0x0000_0010	<a href="#">14.4.6.4/14-65</a>
0x450	PCI Express Power Management Timer Register (PEX_PM_TIMER)	Mixed	0x000A_63E4	<a href="#">14.4.6.5/14-66</a>
0x454	PCI Express PME Time-Out Register (PEX_PME_TIMEOUT)	Mixed	0x00FD_4BC0	<a href="#">14.4.6.6/14-67</a>
0x45C	PCI Express ASPM Request Timer Register (PEX_ASPM_REQTMR) (RC mode only)	R/W	0x0000_0629	<a href="#">14.4.6.7/14-67</a>
0x478	PCI Express Subsystem Vendor ID Update Register (PEX_SSVID_UPDATE)	R/W	0x0000_0000	<a href="#">14.4.6.8/14-68</a>
0x47C	PCI Express Device Capabilities Update Register (PEX_DEVCAP_UPDATE)	R/W	0x0000_0000	<a href="#">14.4.6.9/14-68</a>
0x480	PCI Express Link Capabilities Update Register (PEX_LINKCAP_UPDATE)	R/W	0x0000_3D41	<a href="#">14.4.6.10/14-69</a>
0x490	PCI Express Slot Capabilities Update Register (PEX_SLCAP_UPDATE)	R/W	0x0000_07C0	<a href="#">14.4.6.11/14-71</a>
0x4B0	PCI Express Configuration Ready Register (PEX_CFG_READY)	Mixed	0x0000_0000	<a href="#">14.4.6.12/14-71</a>
<b>PCI Express BAR Configuration Registers (EP Mode)</b>				
0x4D8	PCI Express BAR Size Low Configuration Register (PEX_BAR_SIZEL)	R/W	0xFC00_0000	<a href="#">14.4.7.1/14-72</a>
0x4DC	Reserved	—	—	—
0x4E0	PCI Express BAR Select Configuration Register (PEX_BAR_SEL)	R/W	0x0000_0400	<a href="#">14.4.7.2/14-73</a>

Table 14-3. PCI Express Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
<b>PCI Express—Block Base Address 0x0_9000</b>				
0x504	PCI Express BAR Prefetch Configuration Register (PEX_BAR_PF)	R/W	0x0000_0400	<a href="#">14.4.7.3/14-73</a>
<b>PCI Express Extended Status and Control Register</b>				
0x590	PCI Express PME_To_Ack Timeout Register (PEX_PME_TO_ACK_TOR)	Mixed	0x0019_5460	<a href="#">14.4.8.1/14-74</a>
0x594	PCI Express PME_To_Ack Status Register (PEX_PME_TO_ACK_SR)	w1c	0x0000_0000	<a href="#">14.4.8.2/14-75</a>
0x5A0	PCI Express PCI Interrupt Mask Register (PEX_SS_INTR_MASK)	Mixed	0x0000_003F	<a href="#">14.4.8.3/14-76</a>
<b>PCI Express CSB Bridge Registers</b>				
<b>Global Registers</b>				
0x800–0x804	Reserved	—	—	—
0x808	PCI Express CSB Bridge Control register (PEX_CSB_CTRL)	R/W	0x0000_0130	<a href="#">14.5.2.1/14-77</a>
0x80C	Reserved	—	—	—
0x814	PCI Express DMA Descriptor Timer Register (PEX_DMA_DSTMR)	R/W	0x0000_0000	<a href="#">14.5.2.2/14-78</a>
0x818	Reserved	—	—	—
0x81C	PCI Express CSB Bridge Status register (PEX_CSB_STAT)	R	0x0000_0000	<a href="#">14.5.2.3/14-79</a>
0x820	Reserved	—	—	—
<b>PCI Express Outbound PIO Registers</b>				
0x840	PCI Express Outbound PIO Control Register (PEX_CSB_OBCTRL)	R/W	0x0000_0000	<a href="#">14.5.3.1/14-80</a>
0x844	PCI Express Outbound PIO Status Register (PEX_CSB_OBSTAT)	w1c	0x0000_0000	<a href="#">14.5.3.2/14-81</a>
0x848	Reserved	—	—	—
<b>PCI Express Inbound PIO Registers</b>				
0x8E0	PCI Express Inbound PIO Control Register (PEX_CSB_IBCTRL)	R/W	0x0000_0000	<a href="#">14.5.4.1/14-82</a>
0x8E4	PCI Express Inbound PIO Status Register (PEX_CSB_IBSTAT)	w1c	0x0000_0000	<a href="#">14.5.4.2/14-82</a>
0x8E8	Reserved	—	—	—
<b>PCI Express DMA Registers</b>				
0x990	Reserved	—	—	—
0x9A0	PCI Express Write DMA Control Register (PEX_WDMA_CTRL)	R/W	0x0000_0000	<a href="#">14.5.5.1/14-83</a>
0x9A4	PCI Express Write DMA first Address Register (PEX_WDMA_ADDR)	R/W	0x0000_0000	<a href="#">14.5.5.2/14-84</a>
0x9A8	PCI Express Write DMA Status Register (PEX_WDMA_STAT)	w1c	0x0000_0000	<a href="#">14.5.5.3/14-84</a>

Table 14-3. PCI Express Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
<b>PCI Express—Block Base Address 0x0_9000</b>				
0x9AC	Reserved	—	—	—
0xA40	PCI Express Read DMA Control Register (PEX_RDMA_CTRL)	R/W	0x0000_0000	14.5.5.4/14-85
0xA44	PCI Express Read DMA first Address Register (PEX_RDMA_ADDR)	R/W	0x0000_0000	14.5.5.5/14-86
0xA48	PCI Express Read DMA Status Register (PEX_RDMA_STAT)	w1c	0x0000_0000	14.5.5.6/14-86
<b>Mailbox Registers</b>				
0xB20	PCI Express Outbound Mailbox Control Register (PEX_OMBCR)	R/W	0x0000_0000	14.5.6.1/14-87
0xB24	PCI Express Outbound Mailbox Data Register (PEX_OMBDR)	R/W	0x0000_0000	14.5.6.2/14-88
0xB60	PCI Express Inbound Mailbox Control Register (PEX_IMBCR)	R/W	0x0000_0000	14.5.6.3/14-88
0xB64	PCI Express Inbound Mailbox Data Register (PEX_IMBDR)	R/W	0x0000_0000	14.5.6.4/14-89
<b>PCI Express Host Interrupts Registers</b>				
0xBA0	PCI Express Host Interrupt Enable Register (PEX_HIER)	R/W	0x0000_0000	14.5.7.1/14-89
0xBA4	PCI Express Host Interrupt Status Register (PEX_HISR)	w1c	0x0000_0000	14.5.7.2/14-90
0xBA8	PCI Express Host Outbound PIO Interrupt Vector Register (PEX_HOPIVR)	R/W	0x0000_0000	14.5.7.3/14-91
0xBC0	PCI Express Host Inbound PIO Interrupt Vector Register (PEX_HIPIVR)	R/W	0x0000_0000	14.5.7.4/14-92
0xBC8	PCI Express Host Write DMA Interrupt Vector Register (PEX_HWDIVR)	R/W	0x0000_0000	14.5.7.5/14-92
0xBD0	PCI Express Host Read DMA Interrupt Vector Register (PEX_HRDIVR)	R/W	0x0000_0000	14.5.7.6/14-93
0xBD8	PCI Express Host Miscellaneous Interrupt Vector Register (PEX_HMIVR)	R/W	0x0000_0000	14.5.7.7/14-93
<b>CSB System Interrupts Registers</b>				
0xBE0	CSB System PIO Interrupt Enable Register (PEX_CSPIER)	R/W	0x0000_0000	14.5.8.1/14-94
0xBE4	CSB System Write DMA Interrupt Enable Register (PEX_CSWDIER)	R/W	0x0000_0000	14.5.8.2/14-94
0xBE8	CSB System Read DMA Interrupt Enable Register (PEX_CSRDIER)	R/W	0x0000_0000	14.5.8.3/14-95
0xBEC	CSB System Miscellaneous Interrupt Enable Register (PEX_CSMIER)	R/W	0x0000_0002	14.5.8.4/14-96
0xBF0	CSB System PIO Interrupt Status Register (PEX_CSPISR)	w1c	0x0000_0000	14.5.8.5/14-98
0xBF4	CSB System Write DMA Interrupt Status Register (PEX_CSWDISR)	w1c	0x0000_0000	14.5.8.6/14-99
0xBF8	CSB System Read DMA Interrupt Status Register (PEX_CSRDISR)	w1c	0x0000_0000	14.5.8.7/14-99

Table 14-3. PCI Express Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
<b>PCI Express—Block Base Address 0x0_9000</b>				
0xBFC	CSB System Miscellaneous Interrupt Status Register (PEX_CSMISR)	w1c	0x0000_0000	14.5.8.8/14-100
<b>Power Management Registers</b>				
0xC80	PCI Express PM Control Register (PEX_PM_CTRL)	R/W	0x0000_0000	14.5.9.1/14-102
<b>PCI Express Outbound Address Mapping Registers</b>				
0xCA0	PCI Express Outbound Window Attributes Register 0 (PEX_OWAR0)	R/W	0x0000_0000	14.5.10.1/14-103
0xCA4	PCI Express Outbound Window Base Address Register 0 (PEX_OWBAR0)	R/W	0x0000_0000	14.5.10.2/14-104
0xCA8	PCI Express Outbound Window Translation Address Register Low 0 (PEX_OWTARL0)	R/W	0x0000_0000	14.5.10.3/14-105
0xCAC	PCI Express Outbound Window Translation Address Register High 0 (PEX_OWTARH0)	R/W	0x0000_0000	14.5.10.4/14-105
0xCB0	PCI Express Outbound Window Attributes Register 1 (PEX_OWAR1)	R/W	0x0000_0000	14.5.10.1/14-103
0xCB4	PCI Express Outbound Window Base Address Register 1 (PEX_OWBAR1)	R/W	0x0000_0000	14.5.10.2/14-104
0xCB8	PCI Express Outbound Window Translation Address Register Low 1 (PEX_OWTARL1)	R/W	0x0000_0000	14.5.10.3/14-105
0CBC	PCI Express Outbound Window Translation Address Register High 1 (PEX_OWTARH1)	R/W	0x0000_0000	14.5.10.4/14-105
0xCC0	PCI Express Outbound Window Attributes Register 2 (PEX_OWAR2)	R/W	0x0000_0000	14.5.10.1/14-103
0xCC4	PCI Express Outbound Window Base Address Register 2 (PEX_OWBAR2)	R/W	0x0000_0000	14.5.10.2/14-104
0xCC8	PCI Express Outbound Window Translation Address Register Low 2 (PEX_OWTARL2)	R/W	0x0000_0000	14.5.10.3/14-105
0CCC	PCI Express Outbound Window Translation Address Register High 2 (PEX_OWTARH2)	R/W	0x0000_0000	14.5.10.4/14-105
0xCD0	PCI Express Outbound Window Attributes Register 3 (PEX_OWAR3)	R/W	0x0000_0000	14.5.10.1/14-103
0xCD4	PCI Express Outbound Window Base Address Register 3 (PEX_OWBAR3)	R/W	0x0000_0000	14.5.10.2/14-104
0xCD8	PCI Express Outbound Window Translation Address Register Low 3 (PEX_OWTARL3)	R/W	0x0000_0000	14.5.10.3/14-105
0CDC	PCI Express Outbound Window Translation Address Register High 3 (PEX_OWTARH3)	R/W	0x0000_0000	14.5.10.4/14-105
<b>PCI Express EP Inbound Address Translation Registers</b>				
0xDE0	PCI Express EP Inbound Window Translation Address Register 0 (PEX_EPIWTAR0)	R/W	0x0000_0000	14.5.11.1/14-107

Table 14-3. PCI Express Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
<b>PCI Express—Block Base Address 0x0_9000</b>				
0xDE4	PCI Express EP Inbound Window Translation Address Register 1 (PEX_EPIWTAR1)	R/W	0x0000_0000	<a href="#">14.5.11.1/14-107</a>
0xDE8	PCI Express EP Inbound Window Translation Address Register 2 (PEX_EPIWTAR2)	R/W	0x0000_0000	<a href="#">14.5.11.1/14-107</a>
0xDEC	PCI Express EP Inbound Window Translation Address Register 3 (PEX_EPIWTAR3)	R/W	0x0000_0000	<a href="#">14.5.11.1/14-107</a>
<b>PCI Express RC Inbound Address Mapping Registers</b>				
0xE60	PCI Express RC Inbound Window Attributes Register 0 (PEX_RCIWAR0)	R/W	0x0000_0000	<a href="#">14.5.12.1/14-108</a>
0xE64	PCI Express RC Inbound Window Translation Address Register 0 (PEX_RCIWTAR0)	R/W	0x0000_0000	<a href="#">14.5.12.2/14-109</a>
0xE68	PCI Express RC Inbound Window Base Address Register Low 0 (PEX_RCIWBARL0)	R/W	0x0000_0000	<a href="#">14.5.12.3/14-109</a>
0xE6C	PCI Express RC Inbound Window Base Address Register High 0 (PEX_RCIWBARH0)	R/W	0x0000_0000	<a href="#">14.5.12.4/14-110</a>
0xE70	PCI Express RC Inbound Window Attributes Register 1 (PEX_RCIWAR1)	R/W	0x0000_0000	<a href="#">14.5.12.1/14-108</a>
0xE74	PCI Express RC Inbound Window Translation Address Register 1 (PEX_RCIWTAR1)	R/W	0x0000_0000	<a href="#">14.5.12.2/14-109</a>
0xE78	PCI Express RC Inbound Window Base Address Register Low 1 (PEX_RCIWBARL1)	R/W	0x0000_0000	<a href="#">14.5.12.3/14-109</a>
0xE7C	PCI Express RC Inbound Window Base Address Register High 1 (PEX_RCIWBARH1)	R/W	0x0000_0000	<a href="#">14.5.12.4/14-110</a>
0xE80	PCI Express RC Inbound Window Attributes Register 2 (PEX_RCIWAR2)	R/W	0x0000_0000	<a href="#">14.5.12.1/14-108</a>
0xE84	PCI Express RC Inbound Window Translation Address Register 2 (PEX_RCIWTAR2)	R/W	0x0000_0000	<a href="#">14.5.12.2/14-109</a>
0xE88	PCI Express RC Inbound Window Base Address Register Low 2 (PEX_RCIWBARL2)	R/W	0x0000_0000	<a href="#">14.5.12.3/14-109</a>
0xE8C	PCI Express RC Inbound Window Base Address Register High 2 (PEX_RCIWBARH2)	R/W	0x0000_0000	<a href="#">14.5.12.4/14-110</a>
0xE90	PCI Express RC Inbound Window Attributes Register 3 (PEX_RCIWAR3)	R/W	0x0000_0000	<a href="#">14.5.12.1/14-108</a>
0xE94	PCI Express RC Inbound Window Translation Address Register 3 (PEX_RCIWTAR3)	R/W	0x0000_0000	<a href="#">14.5.12.2/14-109</a>
0xE98	PCI Express RC Inbound Window Base Address Register Low 3 (PEX_RCIWBARL3)	R/W	0x0000_0000	<a href="#">14.5.12.3/14-109</a>
0xE9C	PCI Express RC Inbound Window Base Address Register High 3 (PEX_RCIWBARH3)	R/W	0x0000_0000	<a href="#">14.5.12.4/14-110</a>

<sup>1</sup> MPC8308 does not support these registers in accordance with the PCIe specification. For more information, see *PCI Express Base Specification, March 28, 2005* (Page 357-358). These registers are mentioned here only for completeness. It is recommended not to change the reset values of these registers.

## 14.4 PCI Express Core Configuration Header Registers

The PCI Express core implements a standard type 0/type 1 configuration space, which consists of a 64-byte type 0 configuration space header and capability structures listed in the PCI Express specification.

The various capabilities supported are as follows:

- Power management (PM)
- PCI Express (PCI\_EX)
- Message signaled interrupt (MSI) (not present for RC)
- Vital product data (VPD) (not present for RC)
- Subsystem ID and subsystem vendor ID (SSID/SSVID) (optional capability only for type-1 header devices)

The supported PCI Express extended capabilities are as follows:

- Advanced error reporting
- Device serial number
- Power budgeting VC capability (including VC arbitration table for WRR-32/port arbitration table)
- Vendor specific capability (VSEC)

### 14.4.1 Common PCI Express-Compatible Configuration Header Registers

The first 64 bytes of the 256-byte PCI Express-compatible configuration space consists of a predefined header that every PCI Express-compatible device must support. The first 16 bytes of the predefined header are defined the same way for all PCI Express devices. These common registers are shown in [Figure 14-2](#). They are common to both type 0 and type 1 configuration headers.

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 14-2. PCI Express PCI Express-Compatible Configuration Header Common Registers**

The remaining 48 bytes of the header may have differing layouts depending on the function of the device. Two header types apply to PCI Express. Type 0 headers, described in [Section 14.4.2, “Type 0 PCI Express-Compatible Configuration Header Registers,”](#) are typically used by endpoints; Type 1 headers described in [Section 14.4.3, “Type 1 PCI-Compatible Configuration Header Registers,”](#) are used by root complexes and switches/bridges.

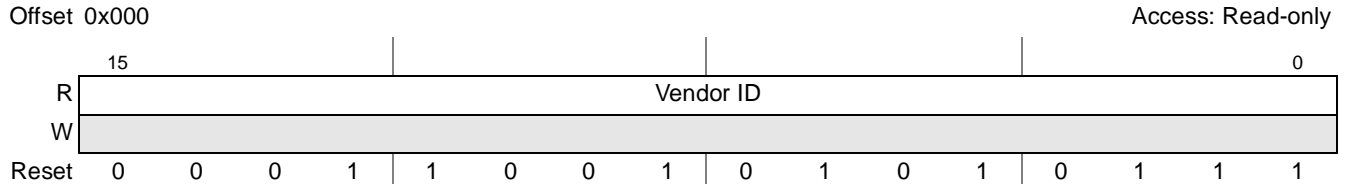


**NOTE**

The registers described in this section use little-endian byte ordering. Software running on the local processor in big-endian mode must byte-swap the data. No byte swapping occurs when the registers are accessed from the PCI Express bus.

**14.4.1.1 PCI Express Vendor ID Register**

The vendor ID register, shown in [Figure 14-3](#), identifies the manufacturer of the device.

**Figure 14-3. PCI Express Vendor ID Register**

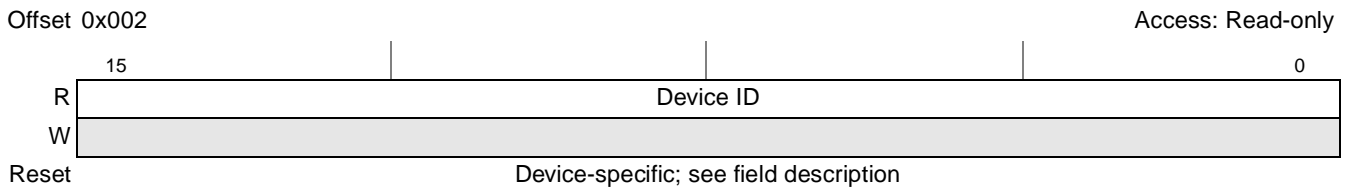
[Table 14-4](#) describes the vendor ID register fields.

**Table 14-4. PCI Express Vendor ID Register Field Description**

Bits	Name	Description
15–0	Vendor ID	0x1957 (Freescale)

**14.4.1.2 PCI Express Device ID Register**

The device ID register, shown in [Figure 14-4](#), identifies the device.

**Figure 14-4. PCI Express Device ID Register**

[Table 14-5](#) describes the device ID register fields.

**Table 14-5. PCI Express Device ID Register Field Description**

Bits	Name	Description
15–0	Device ID	Device ID. This field identifies the device. C006 MPC8308

**14.4.1.3 PCI Express Command Register**

The PCI Express command register, shown in [Figure 14-5](#), controls the ability to generate and respond to PCI Express cycles. The error control and status bits in the command and status registers control PCI Express-compatible error reporting. Note that PCI Express advanced error reporting is controlled by

the PCI Express device control register described in Section 14.4.4.10, “PCI Express Device Control Register,” and the advance error reporting capability structure described in Section 14.4.5.1, “PCI Express Advanced Error Reporting Capability ID Register,” through Section 14.4.5.11, “PCI Express Error Source Identification Register.”

Offset 0x004

Access: Mixed

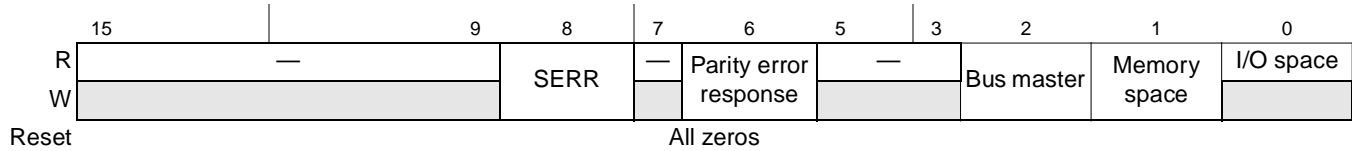


Figure 14-5. PCI Express Command Register

Table 14-6 describes the bits of the command register.

Table 14-6. PCI Express Command Register Fields Description

Bits	Name	Description
15–9	—	Reserved
8	SERR	Controls the reporting of fatal and non-fatal errors detected by the device to the root complex. 0 Disables reporting 1 Enables reporting
7	—	Reserved
6	Parity error response	Controls whether this PCI Express controller responds to parity errors. 0 Parity errors are ignored and normal operation continues. 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.
5–3	—	Reserved
2	Bus master	Enables/disables this PCI Express device to behave as a PCI Express bus master. 0 Disables the ability to generate PCI Express accesses. 1 Enables this PCI Express controller to behave as a bus master. 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.
1	Memory space	Controls whether this PCI Express device (as a target) responds to memory accesses. 0 Device does not respond to PCI Express memory space accesses. 1 Device responds to PCI Express memory space accesses. Clearing this bit prevents the device from accepting any memory transaction. It does not affect outbound memory transactions.
0	I/O space	I/O space. This bit is hardwired to 0.

### 14.4.1.4 PCI Express Status Register

The status register, shown in Figure 14-6, records status information for PCI Express events.

Offset 0x006

Access: Mixed

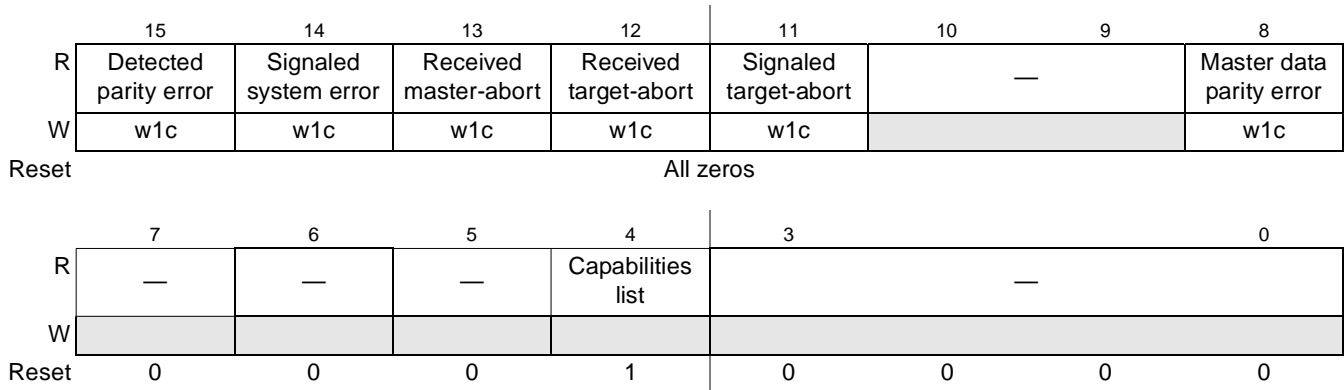


Figure 14-6. PCI Express Status Register

Table 14-7 describes the PCI Express status register bits.

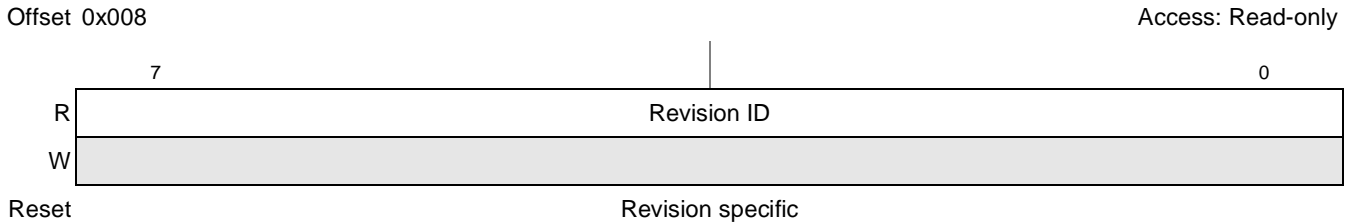
Table 14-7. PCI Express Status Register Fields Description

Bits	Name	Description
15	Detected parity error <sup>1</sup>	Set when a device receives a poisoned TLP regardless of the state of bit 6 in the command register.
14	Signaled system error <sup>1</sup>	Set when 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 <sup>1</sup>	Set when a requestor receives a completion with unsupported request completion status.
12	Received target-abort <sup>1</sup>	Set when a device receives a completion with completer abort completion status.
11	Signaled target-abort <sup>1</sup>	Set when a device completes a request using completer abort completion status.
10–9	—	Reserved
8	Master data parity error detected <sup>1</sup>	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 before this bit can be set.
7–5	—	Reserved.
4	Capabilities List	All PCI Express devices are required to implement the PCI Express capability structure.
3–0	—	Reserved.

<sup>1</sup> The error control and status bits in the command and status registers control PCI Express-compatible error reporting. PCI Express advanced error reporting is controlled by the PCI Express device control register described in Section 14.4.4.10, “PCI Express Device Control Register,” and the advance error reporting capability structure described in Section 14.4.5.1, “PCI Express Advanced Error Reporting Capability ID Register,” through Section 14.4.5.11, “PCI Express Error Source Identification Register.”

### 14.4.1.5 PCI Express Revision ID Register

The revision ID register, shown in [Figure 14-7](#), identifies the revision of the device.



**Figure 14-7. PCI Express Revision ID Register**

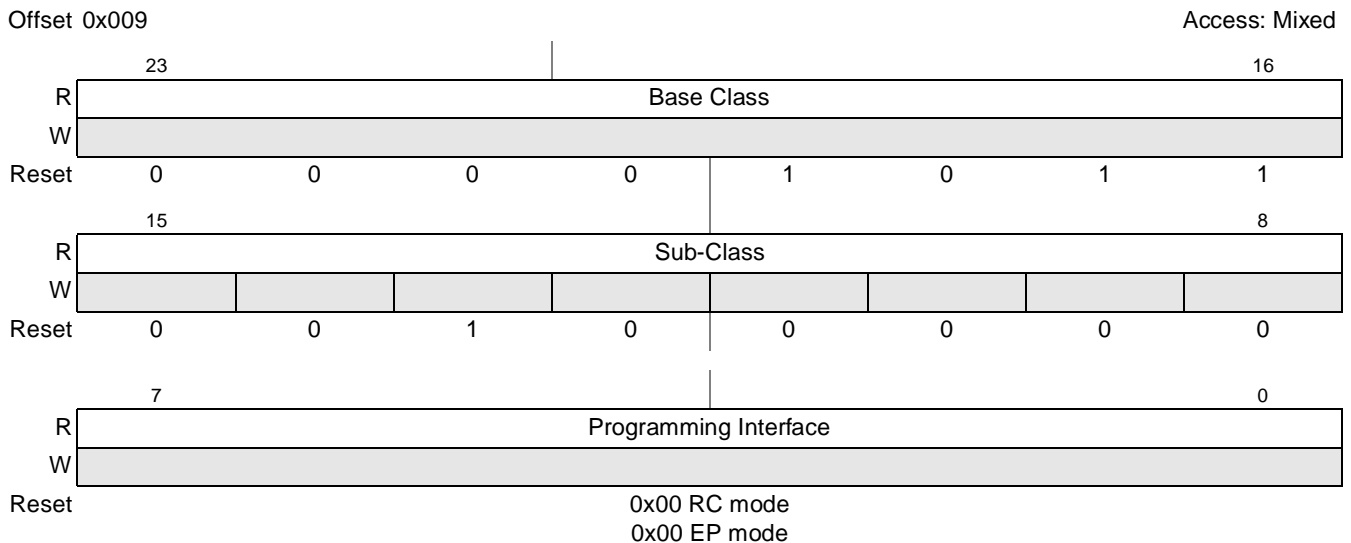
[Table 14-8](#) describes the revision ID register fields.

**Table 14-8. PCI Express Revision ID Register Fields Description**

Bits	Name	Description
7–0	Revision ID	Revision specific. The value is 0x10.

### 14.4.1.6 PCI Express Class Code Register

The PCI Express class code register, shown in [Figure 14-8](#), is composed of three single-byte fields—base class (offset 0x00B), subclass (offset 0x00A), and programming interface (offset 0x009)—that indicate the basic functionality.



**Figure 14-8. PCI Express Class Code Register**

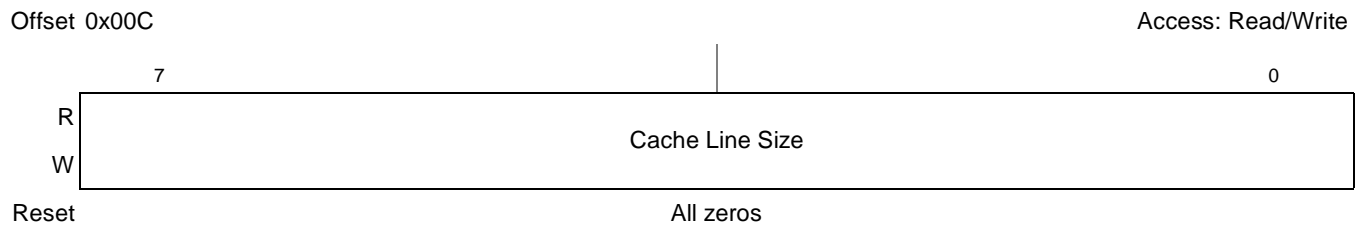
Table 14-9 describes the class code register fields.

**Table 14-9. PCI Express Class Code Register Fields Description**

Bits	Name	Description
23–16	Base Class	0x0B—Processor
15–8	Subclass	0x20—PowerPC
7–0	Programming Interface	0x00—RC mode 0x00—EP mode

### 14.4.1.7 PCI Express Cache Line Size Register

The cache line size register, shown in Figure 14-9, is provided for legacy compatibility (PCI 2.3); it is not used for PCI Express device functionality.



**Figure 14-9. PCI Express Bus Cache Line Size Register**

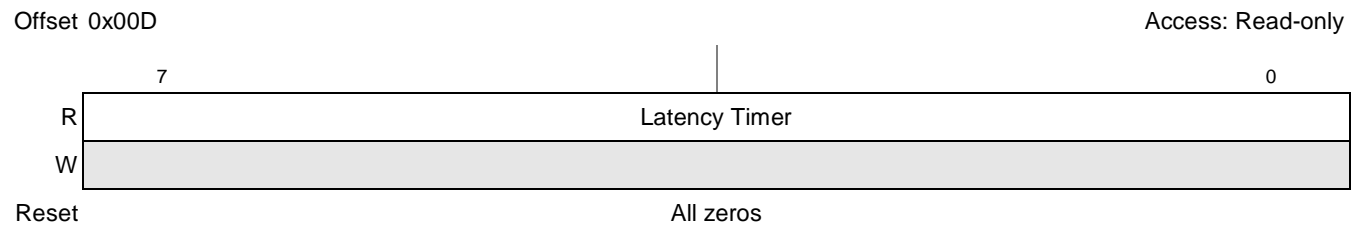
Table 14-10 describes the cache line size register.

**Table 14-10. PCI Express Bus Cache Line Size Register Fields Description**

Bits	Name	Description
7–0	Cache Line Size	Represents the cache line size of the processor in terms of 32-bit words (eight 32-bit words = 32 bytes). Note that for PCI Express operation this register is ignored.

### 14.4.1.8 PCI Express Latency Timer Register

The latency timer register, shown in Figure 14-10, is provided for legacy compatibility (PCI 2.3); it is not used for PCI Express device functionality.



**Figure 14-10. PCI Express Latency Timer Register**

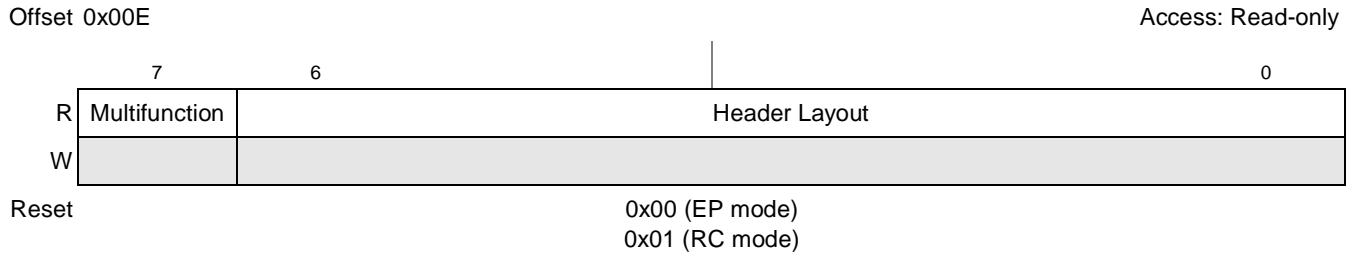
Table 14-11 describes the PCI Express latency timer register (PLTR).

**Table 14-11. PCI Express Latency Timer Register Fields Description**

Bits	Name	Description
7-0	Latency Timer	Note that for PCI Express operation this register is ignored.

### 14.4.1.9 PCI Express Header Type Register

The PCI Express header type register, shown in Figure 14-11, identifies the layout of the PCI Express-compatible header.



**Figure 14-11. PCI Express Header Type Register**

Table 14-12 describes the PCI Express header type register.

**Table 14-12. PCI Express Header Type Register Fields Description**

Bits	Name	Description
7	Multifunction	Identifies whether a device supports multiple functions 0 Single-function device 1 Multiple-function device
6-0	Header Layout	0x00 Endpoint. See Figure 14-12 for type 0 layout. 0x01 Root Complex. See Figure 14-22 for type 1 layout. All other encodings are reserved.

### 14.4.1.10 PCI Express BIST Register

The BIST register is optional and reserved on the PCI Express controller.

## 14.4.2 Type 0 PCI Express-Compatible Configuration Header Registers

The type 0 header is shown in [Figure 14-12](#).

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 14-12. PCI Express PCI Express-Compatible Configuration Header—Type 0**

[Section 14.4.1, “Common PCI Express-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 0x010.

### 14.4.2.1 PCI Express Base Address Registers (EP Mode Only)

The PCI Express base address registers (BARs) point to the beginning of distinct address ranges which the device should claim. The device supports two 32-bit memory space BARs and two 64-bit memory space BARs. These registers in the header configuration space are used for inbound PCI Express transactions, in EP mode only. Note that in RC mode, the device only supports BARs defined by the inbound ATMUs.

For a standard enumeration sequence, the base address registers (BARs) registers are accessed by the Root Complex device to determine the EP attributes. The EP local host should initialize these attributes (if different than the default values) before accepting configuration accesses. This BAR’s size and prefetch

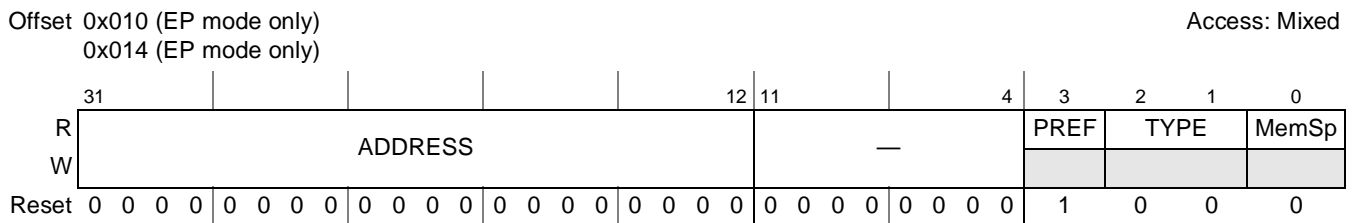
attributes are programmed by an indirect registers access, using the PCI Express BAR Configuration Registers. For further details see [Section 14.4.7, “PCI Express BAR Configuration Registers \(EP Mode\).”](#)

**NOTE**

To access the device internal memory-mapped configuration registers space from the PCI Express side, the IMMRBAR address should be programmed to one of the PCI Express EP inbound window translation address registers (PEX\_EPIWTAR $n$ ) corresponding to one of the base address registers described in this section.

**14.4.2.1.1 Base Address Registers 0 and 1 (BAR0/BAR1)**

BAR0 and BAR1, shown in [Figure 14-13](#), defines the inbound memory windows in the 32-bit memory space.



**Figure 14-13. 32-Bit Base Address Registers (BAR0/BAR1)**

[Table 14-13](#) describes the BAR0/BAR1 fields.

**Table 14-13. BAR0 and BAR1 Register Fields Description**

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 PCI Express BAR configuration registers (EP mode).
11–4	—	Reserved. The device allows a 4 Kbyte window minimum.
3	PREF	Prefetchable. This bit is determined by PCI Express BAR prefetch configuration register (PEX_BAR_PF).
2–1	TYPE	Type. 00 Locate anywhere in 32-bit address space.
0	MemSp	Memory space indicator.



### 14.4.2.1.2 Base Address Registers 2 and 4 (BAR2/BAR4)

BAR2 and BAR4, shown in Figure 14-14, define the lower portion of the 64-bit inbound memory windows.

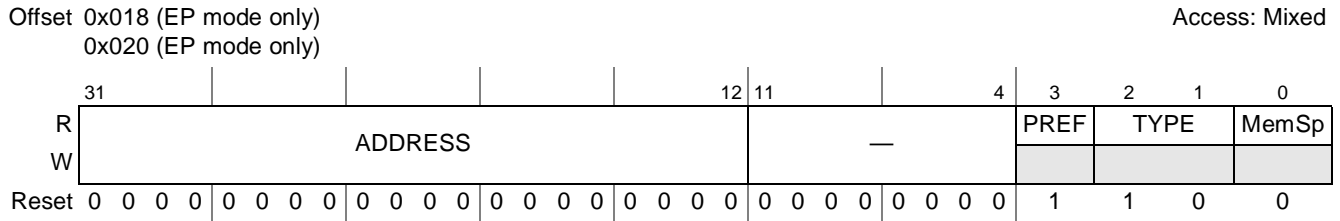


Figure 14-14. 64-Bit Low Memory Base Address Register (BAR2)

Table 14-14 describes the PCI Express 64-bit low memory BAR2 and BAR4 fields.

Table 14-14. BAR2 and BAR4 Register Fields Description

Bits	Name	Description
31–12	ADDRESS	Indicates the lower portion of 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 PCI Express BAR configuration registers (EP mode).
11–4	—	Reserved. The device allows a 4 Kbyte window minimum.
3	PREF	Prefetchable. This bit is determined by PCI Express BAR prefetch configuration register (PEX_BAR_PF).
2–1	TYPE	Type. 0b10 Locate anywhere in 64-bit address space.
0	MemSp	Memory space indicator

### 14.4.2.1.3 Base Address Registers 3 and 5 (BAR3/BAR5)

BAR3/BAR5, shown in Figure 14-15, define the upper portion of the 64-bit inbound memory windows.

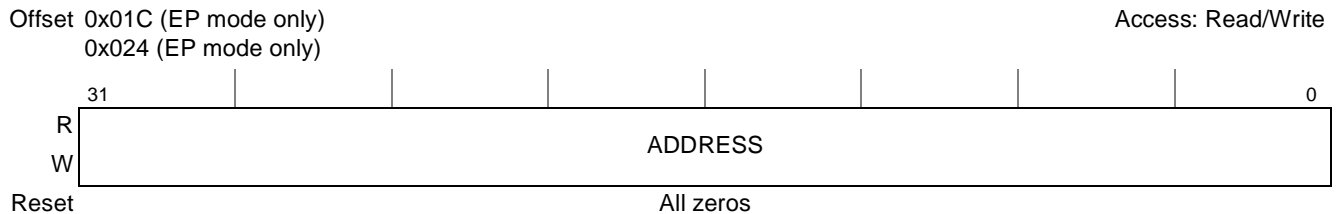


Figure 14-15. 64-Bit High Memory Base Address Registers 3 and 5 (BAR3/BAR5)

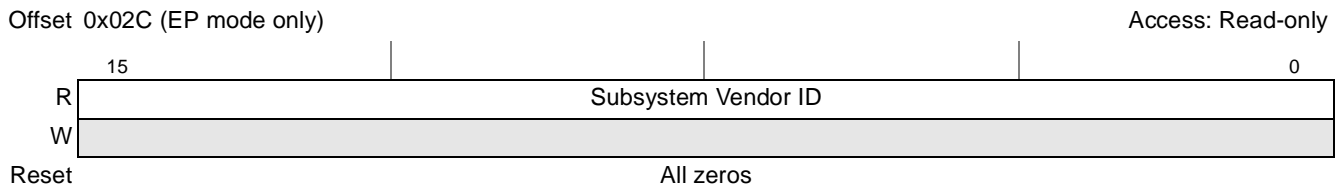
Table 14-15 describes the BAR3 and BAR5 fields.

Table 14-15. BAR3 and BAR5 Register Fields Description

Bits	Name	Description
31–0	ADDRESS	Indicates the upper portion of the base address where the inbound memory window begins. Since the local (CSB) address space of the device is only 32 bits (4 Gbytes), this register is all masked (all ones) when accessed during the enumeration sequence.

### 14.4.2.2 PCI Express Subsystem Vendor ID Register (EP Mode Only)

The PCI Express subsystem vendor ID register, shown in [Figure 14-16](#), identifies the subsystem.



**Figure 14-16. PCI Express Subsystem Vendor ID Register**

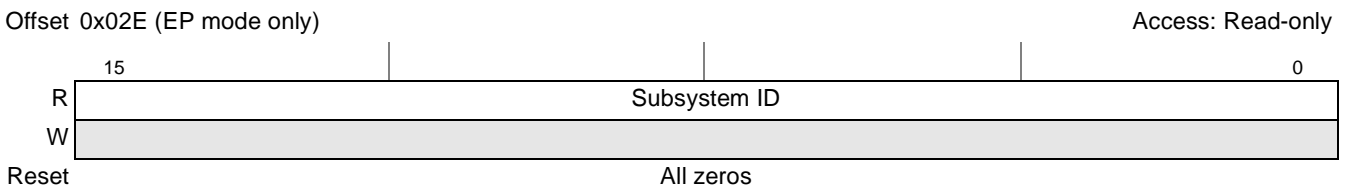
[Table 14-16](#) describes the PCI Express subsystem vendor ID register.

**Table 14-16. PCI Express Subsystem Vendor ID Register Fields Description**

Bits	Name	Description
15–0	Subsystem Vendor ID	Subsystem Vendor ID. The value of this register can be set by programming the SSVID field of the PCI Express Subsystem Vendor ID Update Register (PEX_SSVID_UPDATE). See <a href="#">Section 14.4.6.8, “PCI Express Subsystem Vendor ID Update Register (PEX_SSVID_UPDATE).”</a> This value has to be programmed before setting the config-ready bit in the PCI Express Configuration Ready Register so that the host reads the correct information during enumeration.

### 14.4.2.3 PCI Express Subsystem ID Register (EP Mode Only)

The PCI Express subsystem ID register, shown in [Figure 14-17](#), identifies the subsystem.



**Figure 14-17. PCI Express Subsystem ID Register**

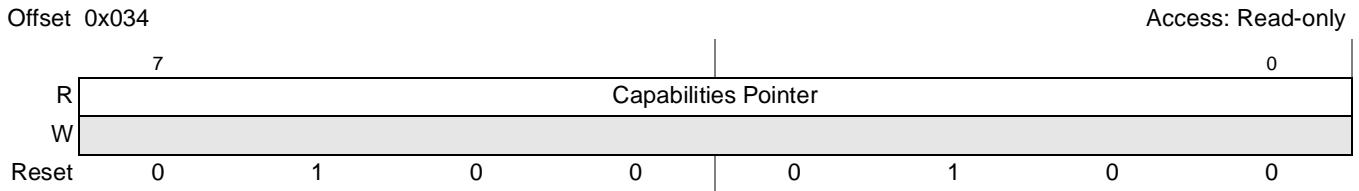
[Table 14-17](#) describes the PCI Express subsystem ID register.

**Table 14-17. PCI Express Subsystem ID Register Fields Description**

Bits	Name	Description
15–0	Subsystem ID	Subsystem ID. The value of this register can be set by programming the SSID field of the PCI Express Subsystem Vendor ID Update Register (PEX_SSVID_UPDATE). See <a href="#">Section 14.4.6.8, “PCI Express Subsystem Vendor ID Update Register (PEX_SSVID_UPDATE).”</a> This value has to be programmed before setting the config-ready bit in the PCI Express Configuration Ready Register so that the host reads the correct information during enumeration.

### 14.4.2.4 PCI Express Capabilities Pointer Register

The PCI Express capabilities pointer register, shown in [Figure 14-18](#), identifies additional functionality supported by the device.



**Figure 14-18. PCI Express Capabilities Pointer Register**

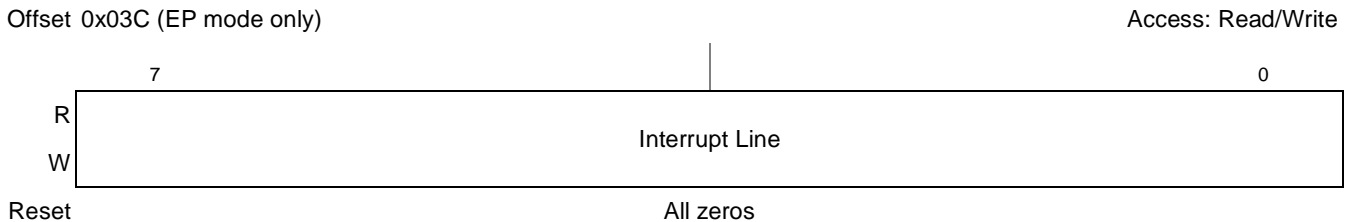
[Table 14-18](#) describes the PCI Express capabilities pointer.

**Table 14-18. PCI Express Capabilities Pointer Register Fields Description**

Bits	Name	Description
7–0	Capabilities Pointer	The capabilities pointer provides the offset (0x44) for additional PCI Express-compatible registers above the common 64-byte header. Refer to <a href="#">Section 14.4.4</a> , “PCI Express-Compatible Device-Specific Configuration Space Registers.”

### 14.4.2.5 PCI Express Interrupt Line Register (EP-Mode Only)

The PCI Express interrupt line register, shown in [Figure 14-19](#), 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 14-19. PCI Express Interrupt Line Register**

[Table 14-19](#) describes the PCI Express interrupt line register.

**Table 14-19. PCI Express Interrupt Line Register Fields Description**

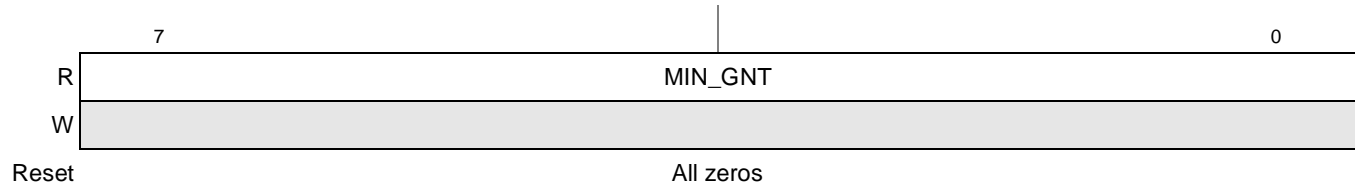
Bits	Name	Description
7–0	Interrupt Line	Communicates interrupt line routing information.

### 14.4.2.6 PCI Express Minimum Grant Register (EP Mode Only)

This register does not apply to PCI Express. It is present for legacy purposes.

Offset 0x03E (EP mode only)

Access: Read-only



**Figure 14-20. PCI Express Minimum Grant Register (MAX\_GNT)**

**Table 14-20. PCI Express Minimum Grant Register Fields Description**

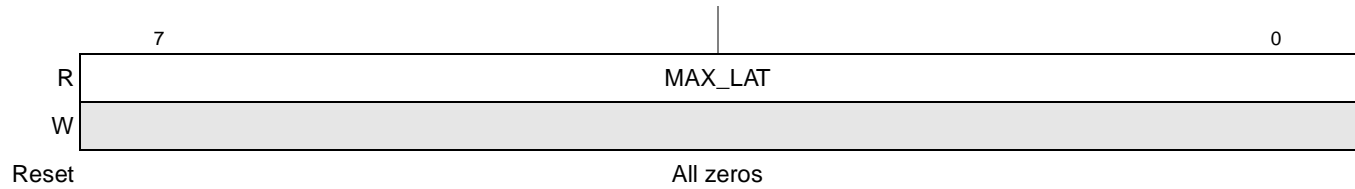
Bits	Name	Description
7-0	MIN_GNT	Does not apply for PCI Express.

### 14.4.2.7 PCI Express Maximum Latency Register (EP Mode Only)

This register does not apply to PCI Express. It is present for legacy purposes.

Offset 0x03F (EP mode only)

Access: Read-only



**Figure 14-21. PCI Express Maximum Latency Register (MAX\_LAT)**

**Table 14-21. PCI Express Maximum Latency Register Fields Description**

Bits	Name	Description
7-0	MAX_LAT	Does not apply for PCI Express.

### 14.4.3 Type 1 PCI-Compatible Configuration Header Registers

The type 1 header is shown in [Figure 14-22](#).

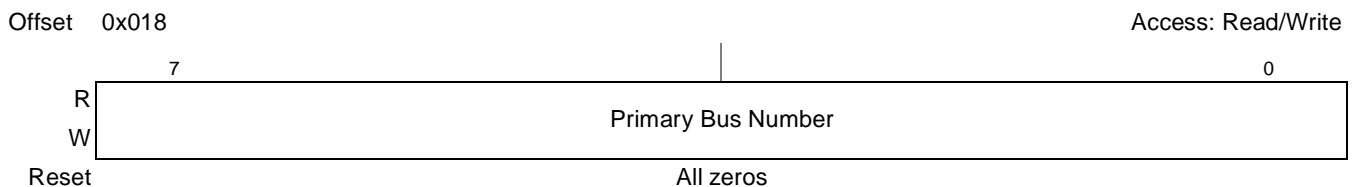
Reserved				Address Offset (Hex)
Device ID		Vendor ID		00
Status		Command		04
Class Code			Revision ID	08
BIST	Header Type	Latency Timer	Cache Line Size	0C
				10
				14
Secondary Latency Timer	Subordinate Bus Number	Secondary Bus Number	Primary Bus Number	18
Secondary Status		I/O Limit	I/O Base	1C
Memory 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
Expansion ROM Base Address				38
Bridge Control		Interrupt Pin	Interrupt Line	3C

**Figure 14-22. PCI Express PCI Express-Compatible Configuration Header—Type 1**

Section 14.4.1, “Common PCI Express-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 0x010.

#### 14.4.3.1 PCI Express Primary Bus Number Register (RC Mode Only)

The primary bus number register is shown in [Figure 14-23](#).



**Figure 14-23. PCI Express Primary Bus Number Register**

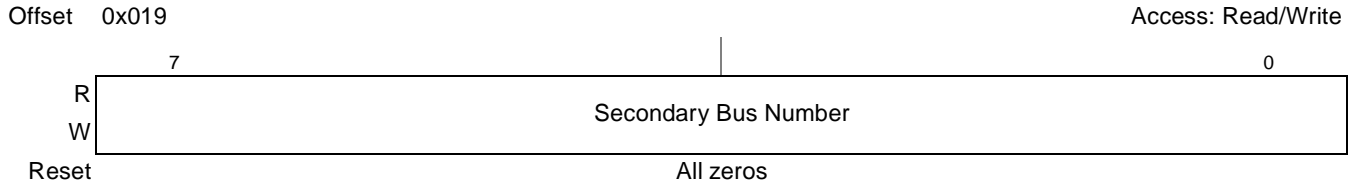
[Table 14-22](#) describes the primary bus number register fields.

**Table 14-22. PCI Express Primary Bus Number Register Fields 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.

### 14.4.3.2 PCI Express Secondary Bus Number Register (RC Mode Only)

The secondary bus number register is shown in [Figure 14-24](#).



**Figure 14-24. PCI Express Secondary Bus Number Register**

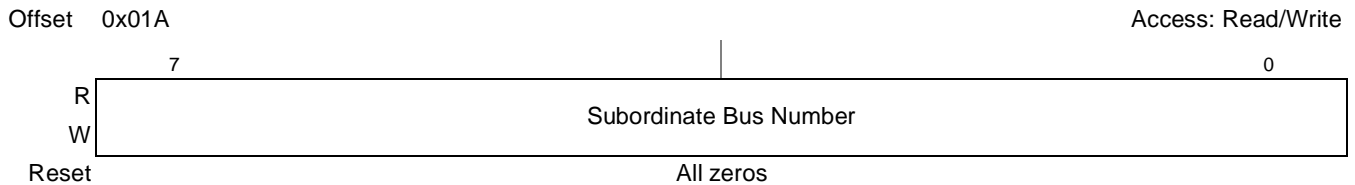
[Table 14-23](#) describes the secondary bus number register fields.

**Table 14-23. PCI Express Secondary Bus Number Register Fields 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.

### 14.4.3.3 PCI Express Subordinate Bus Number Register (RC Mode Only)

The subordinate bus number register is shown in [Figure 14-25](#).



**Figure 14-25. PCI Express Subordinate Bus Number Register**

[Table 14-24](#) describes the subordinate bus number register fields.

**Table 14-24. PCI Express Subordinate Bus Number Register Fields Description**

Bits	Name	Description
7-0	Subordinate Bus Number	Highest bus number that is on the downstream interface.

### 14.4.3.4 PCI Express Secondary Latency Timer Register (RC Mode Only)

The secondary latency timer register does not apply to PCI Express. It must be read-only and return all zeros when read.

### 14.4.3.5 PCI Express I/O Base Register (RC Mode Only)

Note that this device does not support inbound I/O transactions. The I/O base register is shown in Figure 14-26.

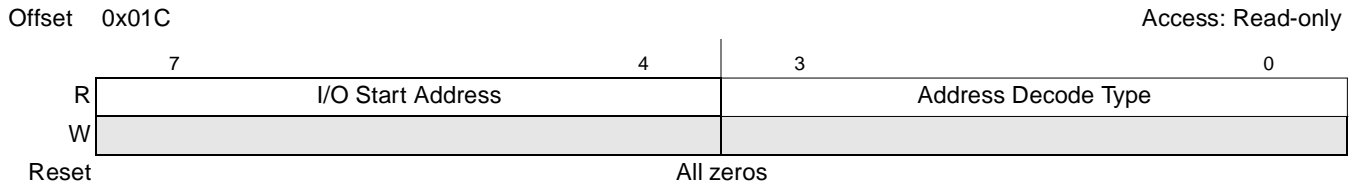


Figure 14-26. PCI Express I/O Base Register

Table 14-25 describes the I/O base register fields.

Table 14-25. PCI Express I/O Base Register Fields 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. 0x00 16-bit I/O address decode 0x01 32-bit I/O address decode All other settings reserved.

### 14.4.3.6 PCI Express I/O Limit Register (RC Mode Only)

Note that this device does not support inbound I/O transactions. The I/O limit register is shown in Figure 14-27.

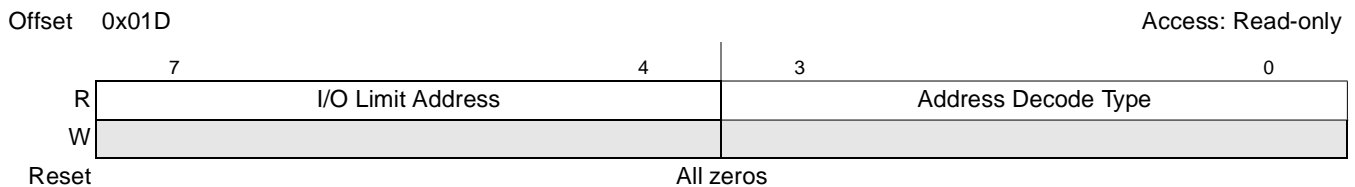


Figure 14-27. PCI Express I/O Limit Register

Table 14-26 describes the I/O limit register fields.

Table 14-26. PCI Express I/O Limit Register Fields 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. 0x00 16-bit I/O address decode 0x01 32-bit I/O address decode All other settings reserved.

### 14.4.3.7 PCI Express Secondary Status Register (RC Mode Only)

The PCI Express secondary status register is shown in Figure 14-28. Note that the errors in this register can be masked by corresponding bits in the secondary status interrupt mask register (PEX\_SS\_INTR\_MASK) and that by default all the errors are masked. See Section 14.4.8.3, “Secondary Status Interrupt Mask Register (PEX\_SS\_INTR\_MASK) (RC Mode Only),” for more information.

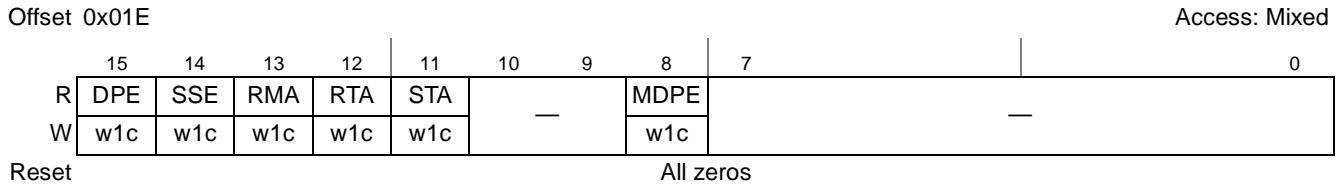


Figure 14-28. PCI Express Secondary Status Register

Table 14-27 describes the PCI Express secondary status register fields.

Table 14-27. PCI Express Secondary Status Register Fields Description

Bits	Name	Description
15	DPE	Detected parity error. This bit is set when 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 if 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

### 14.4.3.8 PCI Express Memory Base Register (RC Mode Only)

The memory base register is shown in Figure 14-29.

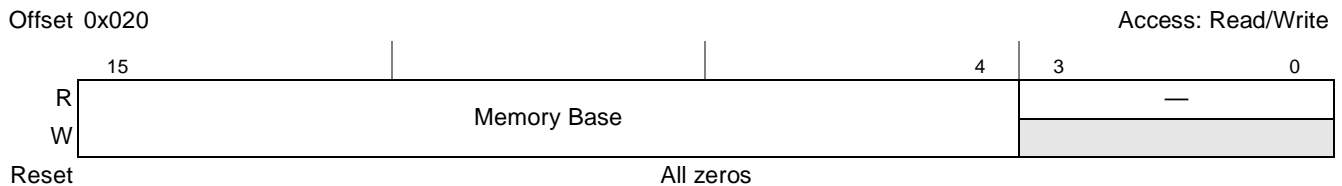


Figure 14-29. PCI Express Memory Base Register



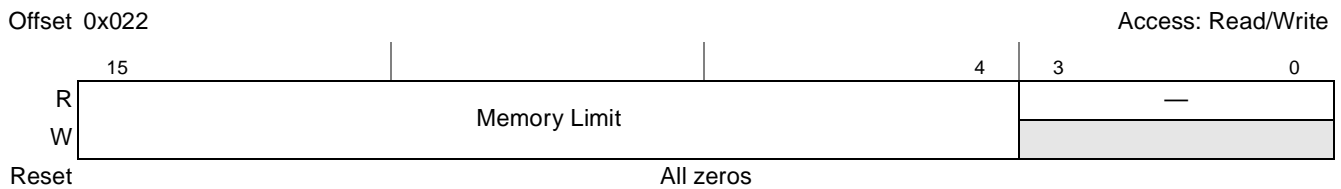
Table 14-28 describes the memory base register fields.

**Table 14-28. PCI Express Memory Base Register Fields 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. <b>Note:</b> 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

### 14.4.3.9 PCI Express Memory Limit Register (RC Mode Only)

The memory limit register is shown in Figure 14-30.



**Figure 14-30. PCI Express Memory Limit Register**

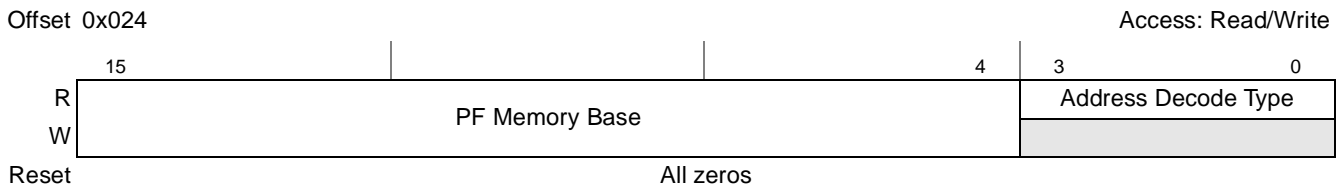
Table 14-29 describes the memory base register fields.

**Table 14-29. PCI Express Memory Limit Register Fields 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. <b>Note:</b> Inbound posted transactions hitting into the mem base/limit range are ignored; inbound non-posted transactions hitting into the mem base/limit range result in an unsupported request response.
3–0	—	Reserved

### 14.4.3.10 PCI Express Prefetchable Memory Base Register (RC Mode Only)

The prefetchable memory base register is shown in Figure 14-31.



**Figure 14-31. PCI Express Prefetchable Memory Base Register**

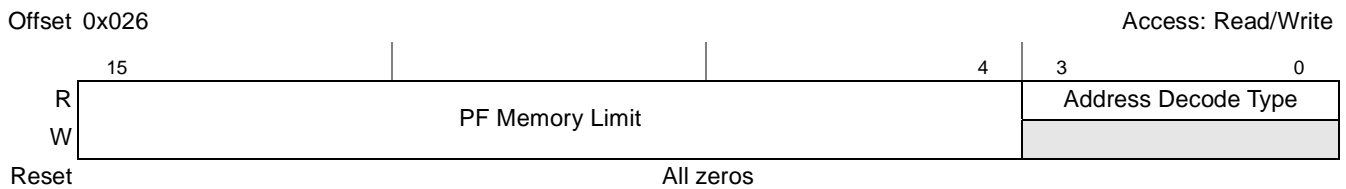
Table 14-30 describes the prefetchable memory base register fields.

**Table 14-30. PCI Express Prefetchable Memory Base Register Fields 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	Number of prefetchable memory address bits. 0x00 32-bit memory address decode 0x01 64-bit memory address decode All other settings reserved.

### 14.4.3.11 PCI Express Prefetchable Memory Limit Register (RC Mode Only)

The PCI Express prefetchable memory limit register is shown in Figure 14-32.



**Figure 14-32. PCI Express Prefetchable Memory Limit Register**

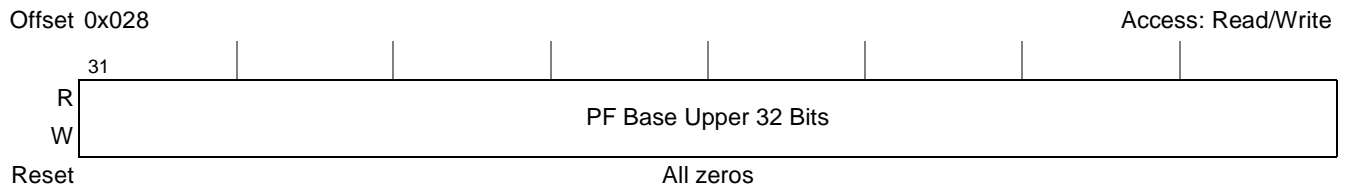
Table 14-31 describes the prefetchable memory limit register fields.

**Table 14-31. PCI Express Prefetchable Memory Limit Register Fields 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. 0x00 32-bit memory address decode 0x01 64-bit memory address decode All other settings reserved.

### 14.4.3.12 PCI Express Prefetchable Base Upper 32-Bit Register (RC Mode Only)

The PCI Express prefetchable memory base upper 32-bit register is shown in Figure 14-33.



**Figure 14-33. PCI Express Prefetchable Base Upper 32-Bit Register**

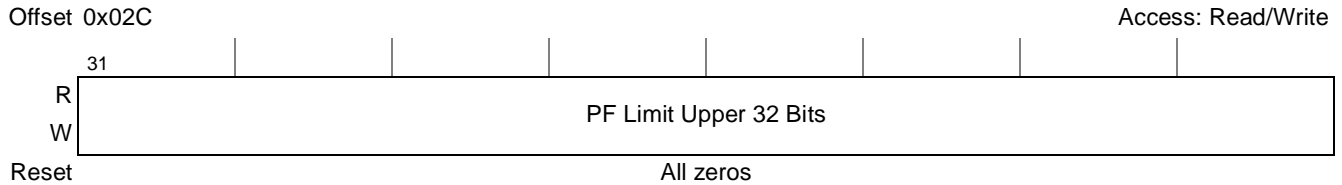
Table 14-32 describes the PCI Express prefetchable memory base upper 32-bit register fields.

**Table 14-32. PCI Express Prefetchable Base Upper 32-Bit Register Fields Description**

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.

### 14.4.3.13 PCI Express Prefetchable Limit Upper 32-Bit Register (RC Mode Only)

The PCI Express prefetchable memory base upper 32-bit register is shown in Figure 14-34.



**Figure 14-34. PCI Express Prefetchable Limit Upper 32-Bit Register**

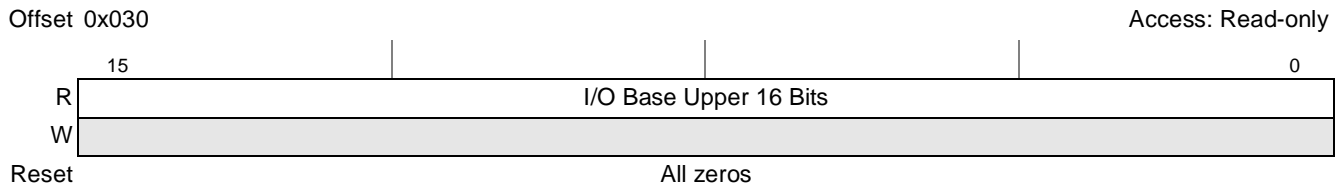
Table 14-33 describes the PCI Express prefetchable memory limit upper 32-bit register fields.

**Table 14-33. PCI Express Prefetchable Limit Upper 32-Bit Register Fields Description**

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.

### 14.4.3.14 PCI Express I/O Base Upper 16-Bit Register (RC Mode Only)

Note that this device does not support inbound I/O transactions. The I/O base upper 16-bit register is shown in Figure 14-35.



**Figure 14-35. PCI Express I/O Base Upper 16-Bit Register**

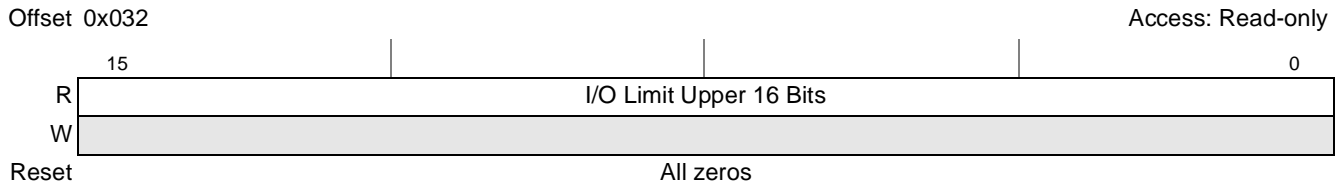
Table 14-34 describes the I/O base upper 16 bits register fields.

**Table 14-34. PCI Express I/O Base Upper 16-Bit Register Fields 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.

### 14.4.3.15 PCI Express I/O Limit Upper 16-Bit Register (RC Mode Only)

Note that this device does not support inbound I/O transactions. The I/O limit upper 16-bit register is shown in [Figure 14-36](#).



**Figure 14-36. PCI Express I/O Limit Upper 16-Bit Register**

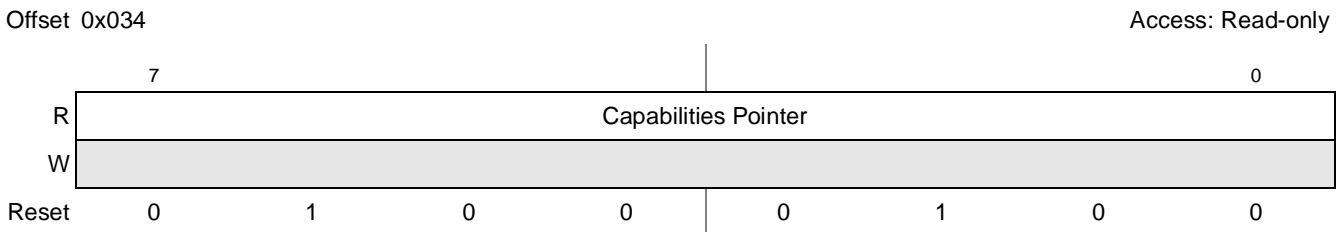
[Table 14-35](#) describes the I/O limit upper 16-bit register fields.

**Table 14-35. PCI Express I/O Limit Upper 16-Bit Register Fields 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.

### 14.4.3.16 PCI Express Capabilities Pointer Register

The PCI Express capabilities pointer, shown in [Figure 14-37](#), identifies additional functionality supported by the device.



**Figure 14-37. PCI Express Capabilities Pointer Register**

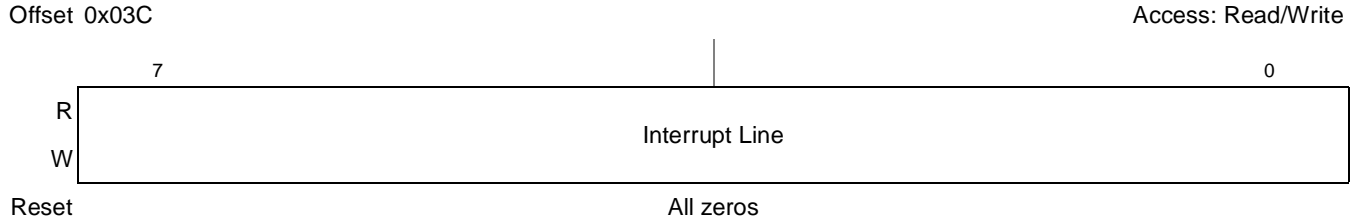
[Table 14-36](#) describes the PCI Express capabilities pointer register fields.

**Table 14-36. PCI Express Capabilities Pointer Register Fields Description**

Bits	Name	Description
7–0	Capabilities Pointer	Provides the offset (0x44) for additional PCI-compatible registers above the common 64-byte header. Refer to <a href="#">Section 14.4.4, “PCI Express-Compatible Device-Specific Configuration Space Registers.”</a>

### 14.4.3.17 PCI Express Interrupt Line Register

The PCI Express interrupt line register, shown in [Figure 14-38](#), 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 14-38. PCI Express Interrupt Line Register**

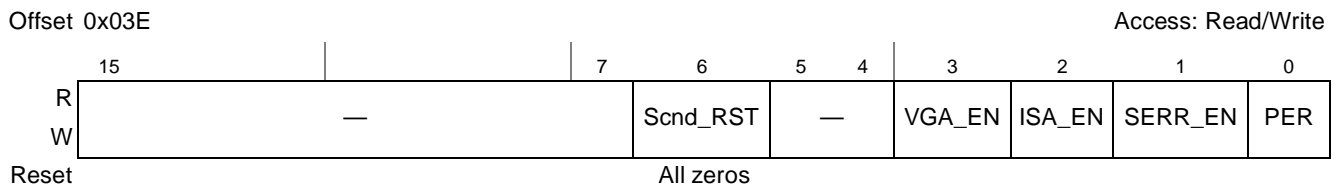
[Table 14-37](#) describes the PCI Express capabilities pointer register fields.

**Table 14-37. PCI Express Interrupt Line Register Fields Description**

Bits	Name	Description
7–0	Interrupt Line	Communicates interrupt line routing information.

### 14.4.3.18 PCI Express Bridge Control Register (RC Mode Only)

The PCI Express bridge control register is shown in [Figure 14-39](#).



**Figure 14-39. PCI Express Bridge Control Register**

[Table 14-38](#) describes the PCI Express bridge control register fields.

**Table 14-38. PCI Express Bridge Control Register Fields 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. Controls the propagation of ERR_COR, ERR_NONFATAL, and ERR_FATAL responses received on the secondary side. If this bit is set and an error message is received from the secondary side, the ERRD bit in PEX_CSMISR is set.
0	PER	Parity error response.

### 14.4.4 PCI Express-Compatible Device-Specific Configuration Space Registers

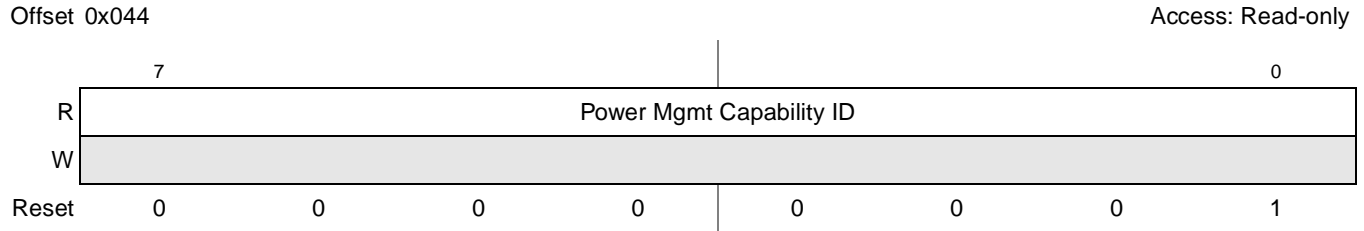
The PCI Express-compatible device-specific configuration space is a PCI Express-compatible configuration space from 0x040 to 0x0FF (just above the 64-byte PCI Express-compatible configuration header).

Reserved	Address Offset (Hex)			
<div style="border: 1px solid black; width: 20px; height: 10px; margin-bottom: 5px;"></div> PCI-Compatible Configuration Header (See Section 14.4.1, “Common PCI Express-Compatible Configuration Header Registers,” for more information.)	000  03F			
	040			
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 40%;">Power Mgmt Capabilities</td> <td style="width: 20%;">Next Pointer (0x4C)</td> <td style="width: 40%;">Power Mgmt Capability ID</td> </tr> </table>	Power Mgmt Capabilities	Next Pointer (0x4C)	Power Mgmt Capability ID	044
Power Mgmt Capabilities	Next Pointer (0x4C)	Power Mgmt Capability ID		
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 25%;">Data</td> <td style="width: 25%;"></td> <td style="width: 50%;">Power Management Status &amp; Control</td> </tr> </table>	Data		Power Management Status & Control	048
Data		Power Management Status & Control		
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 45%;">PCI Express Capabilities</td> <td style="width: 20%;">Next Pointer (0x70 — EP mode) (NULL — RC mode)</td> <td style="width: 35%;">PCI Express Capability ID</td> </tr> </table>	PCI Express Capabilities	Next Pointer (0x70 — EP mode) (NULL — RC mode)	PCI Express Capability ID	04C
PCI Express Capabilities	Next Pointer (0x70 — EP mode) (NULL — RC mode)	PCI Express Capability ID		
Device Capabilities	050			
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 50%;">Device Status</td> <td style="width: 50%;">Device Control</td> </tr> </table>	Device Status	Device Control	054	
Device Status	Device Control			
Link Capabilities	058			
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 50%;">Link Status</td> <td style="width: 50%;">Link Control</td> </tr> </table>	Link Status	Link Control	05C	
Link Status	Link Control			
Slot Capabilities	060			
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 50%;">Slot Status</td> <td style="width: 50%;">Slot Control</td> </tr> </table>	Slot Status	Slot Control	064	
Slot Status	Slot Control			
	068			
Root Status	06C			
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 35%;">MSI Message Control</td> <td style="width: 20%;">Next Pointer (NULL)</td> <td style="width: 45%;">MSI Message Capability ID</td> </tr> </table>	MSI Message Control	Next Pointer (NULL)	MSI Message Capability ID	070
MSI Message Control	Next Pointer (NULL)	MSI Message Capability ID		
MSI Message Address	074			
MSI Upper Message Address	078			
	07C			
	080			
	0FF			

Figure 14-40. PCI Express-Compatible Device-Specific Configuration Space

### 14.4.4.1 PCI Express Power Management Capability ID Register

The PCI Express power management capability ID register is shown in [Figure 14-41](#).



**Figure 14-41. PCI Express Power Management Capability ID Register**

[Table 14-39](#) describes the PCI Express power management capability ID fields.

**Table 14-39. PCI Express Power Management Capability ID Register Fields Description**

Bits	Name	Description
7-0	Power Management Capability ID	Power Management = 0x01

### 14.4.4.2 PCI Express Power Management Next Capabilities Pointer Register

The PCI Express power management next capabilities pointer register is shown in [Figure 14-42](#).



**Figure 14-42. PCI Express Power Management Next Capabilities Pointer**

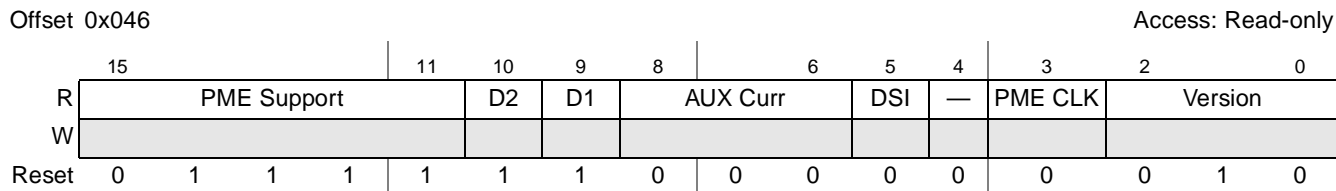
[Table 14-40](#) describes the PCI Express power management next capabilities pointer fields.

**Table 14-40. PCI Express Power Management Next Capabilities Pointer Fields Description**

Bits	Name	Description
7-0	—	Points to the PCI Express Capability Registers

### 14.4.4.3 PCI Express Power Management Capabilities Register

The PCI Express power management capabilities register is shown in [Figure 14-43](#).



**Figure 14-43. PCI Express Power Management Capabilities Register**

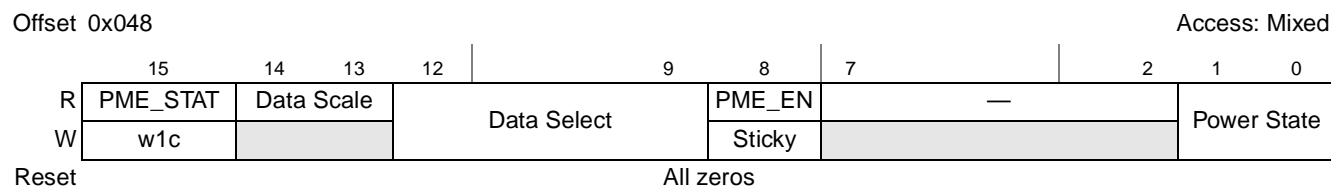
[Table 14-41](#) describes the PCI Express power management capabilities register fields.

**Table 14-41. PCI Express Power Management Capabilities Register Fields Description**

Bits	Name	Description
15–11	PME Support	For a device, this 5-bit field indicates the power states in which the device may generate a PME. PME can be issues from D0, D1, D2 and D3hot.
10	D2	D2 power state is supported.
9	D1	D1 power state is supported.
8–6	AUX Curr	AUX Current. Vaux and D3cold is not supported by this device.
5	DSI	A Device Specific Initialization is not required.
4	—	Reserved
3	PME CLK	Does not apply to PCI Express
2–0	Version	0x02 indicates compatibility to the <i>PCI Express Base Specification, Rev. 1.0a</i>

### 14.4.4.4 PCI Express Power Management Status and Control Register

The PCI Express power management status and control register is shown in [Figure 14-44](#).



**Figure 14-44. PCI Express Power Management Status and Control Register**

[Table 14-42](#) describes the PCI Express power management status and control register fields.

**Table 14-42. PCI Express Power Management Status and Control Register Fields Description**

Bits	Name	Description
15	PME_STAT	PME Status. This bit is set when PME is generated. Writing a “1” to this bit will clear it. Writing a “0” has no effect.
14–13	Data Scale	—
12–9	Data Select	—

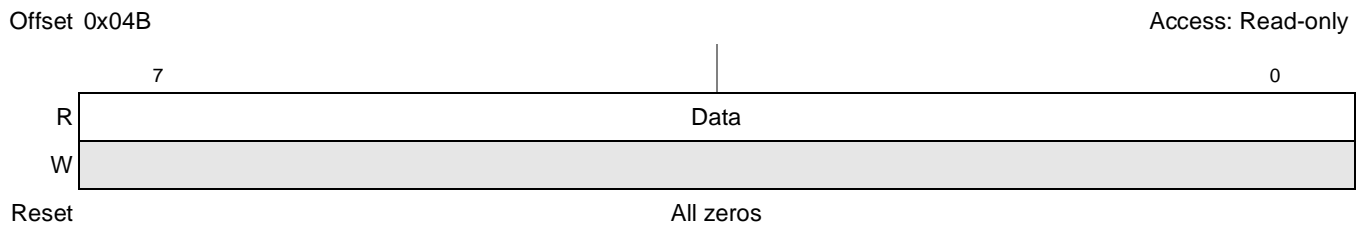


**Table 14-42. PCI Express Power Management Status and Control Register Fields Description (continued)**

Bits	Name	Description
8	PME_EN	PME Enable
7–2	—	Reserved
1–0	Power State	Power state. Indicates the current power state of the function. 00 D0 01 D1 02 D2 03 D3

### 14.4.4.5 PCI Express Power Management Data Register

The PCI Express power management data register is shown in [Figure 14-45](#).



**Figure 14-45. PCI Express Power Management Data Register**

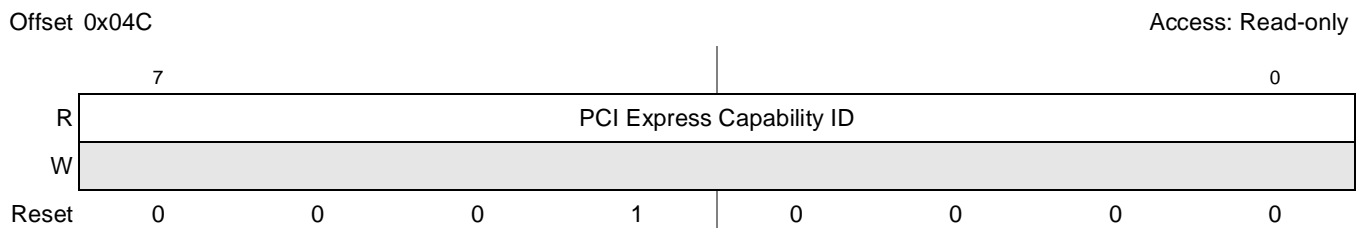
[Table 14-43](#) describes the PCI Express power management data register fields.

**Table 14-43. PCI Express Power Management Data Register Fields Description**

Bits	Name	Description
7–0	Data	—

### 14.4.4.6 PCI Express Capability ID Register

The PCI Express capability ID register is shown in [Figure 14-46](#).



**Figure 14-46. PCI Express Capability ID Register**

Table 14-44 describes the PCI Express capability ID register fields.

**Table 14-44. PCI Express Capability ID Register Fields Description**

Bits	Name	Description
7-0	PCI Express Capability ID	PCI Express = 0x10

### 14.4.4.7 PCI Express Next Capabilities Pointer Register

The PCI Express next capabilities pointer register is shown in Figure 14-47.



1 The reset value of 0b0111\_0000 is only true in EP mode. In RC mode, the reset values should be 0b0000\_0000.

**Figure 14-47. PCI Express Next Capabilities Pointer**

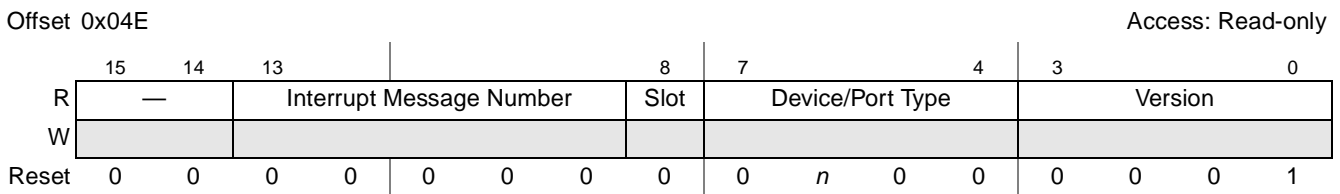
Table 14-45 describes the PCI Express next capabilities pointer fields.

**Table 14-45. PCI Express Next Capabilities Pointer Fields Description**

Bits	Name	Description
7-0		Points to the PCI Express MSI capability registers. The reset value is 0x70 in EP mode and 0x00 in RC mode.

### 14.4.4.8 PCI Express Capabilities Register

The PCI Express capabilities register is shown in Figure 14-48.



**Figure 14-48. PCI Express Capabilities Register**

Table 14-46 describes the PCI Express capabilities register fields.

**Table 14-46. PCI Express Capabilities Register Fields Description**

Bits	Name	Description
15-14	—	Reserved
13-9	Interrupt Message Number	This device supports only a single MSI number.
8	Slot	Slot Implemented (RC mode only)

**Table 14-46. PCI Express Capabilities Register Fields Description (continued)**

Bits	Name	Description
7-4	Device/Port Type	0100 (RC mode) 0000 (EP mode)
3-0	Version	Indicates PCI-SIG defined PCI Express capability structure version number. 0x1 identifies version 1.0a.

#### 14.4.4.9 PCI Express Device Capabilities Register

The PCI Express device capabilities register is shown in Figure 14-49. Note that for End Point mode some of these fields can be set indirectly, using the PCI Express Device Capabilities Update Register. See Section 14.4.6.9, “PCI Express Device Capabilities Update Register (PEX\_DEVCAP\_UPDATE),” for additional details.

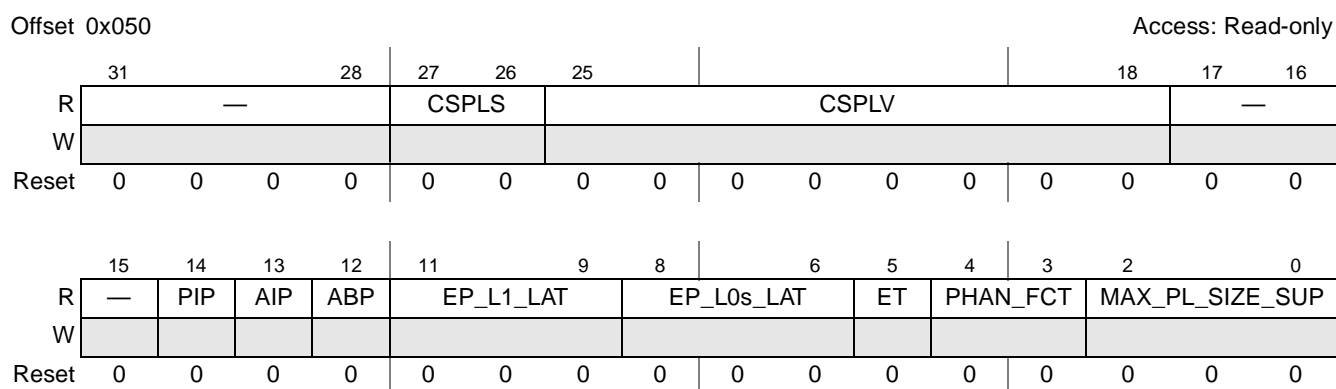

**Figure 14-49. PCI Express Device Capabilities Register**

Table 14-47 describes the PCI Express capabilities register fields.

**Table 14-47. PCI Express Device Capabilities Register Fields 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

**Table 14-47. PCI Express Device Capabilities Register Fields Description (continued)**

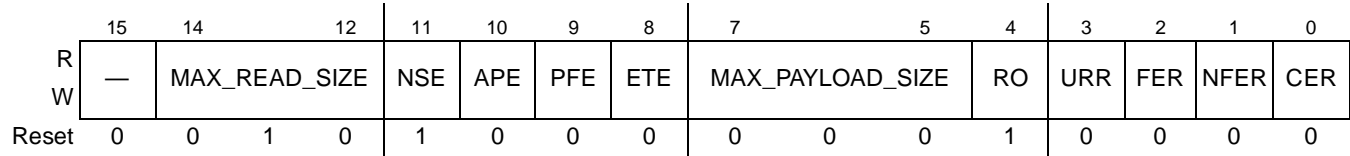
Bits	Name	Description
4-3	PHAN_FCT	Phantom functions supported
2-0	MAX_PL_SIZE_SUP	Maximum payload size supported. 000 = 128 bytes

### 14.4.4.10 PCI Express Device Control Register

The PCI Express device control register is shown in [Figure 14-50](#).

Offset 0x054

Access: Read/write



**Figure 14-50. PCI Express Device Control Register**

[Table 14-48](#) describes the PCI Express device control register fields.

**Table 14-48. PCI Express Device Control Register Fields 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

### 14.4.4.11 PCI Express Device Status Register

The PCI Express device status register is shown in [Figure 14-51](#).

Offset 0x056

Access: Mixed

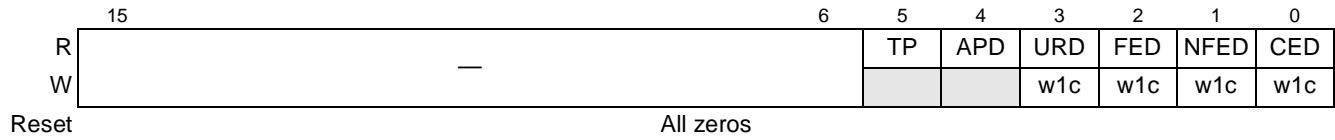


Figure 14-51. PCI Express Device Status Register

[Table 14-49](#) describes the PCI Express device status register fields.

Table 14-49. PCI Express Device Status Register Fields 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

### 14.4.4.12 PCI Express Link Capabilities Register

The PCI Express link capabilities register is shown in [Figure 14-52](#). Note that for End Point mode, some of these fields can indirectly be set using the PCI Express Link Capabilities Update Register (PEX\_LINKCAP\_UPDATE). See [Section 14.4.6.10, “PCI Express Link Capabilities Update Register \(PEX\\_LINKCAP\\_UPDATE\),”](#) for more details.

Offset 0x058

Access: Read-only

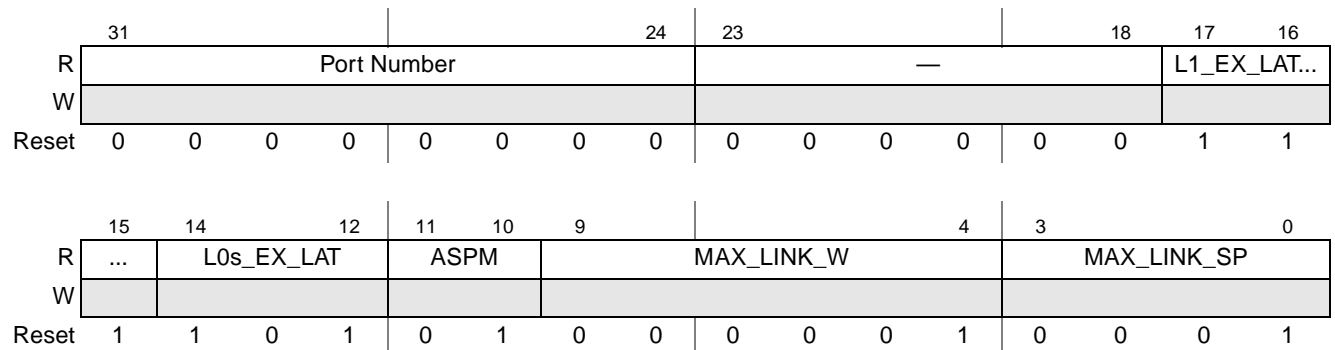


Figure 14-52. PCI Express Link Capabilities Register

Table 14-50 describes the PCI Express link capabilities register fields.

**Table 14-50. PCI Express Link Capabilities Register Fields Description**

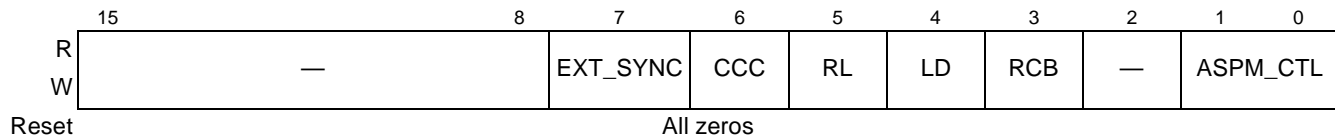
Bits	Name	Description
31–24	Port Number	
23–18	—	Reserved
17–15	L1_EX_LAT	L1 exit latency. 0b111 indicates more than 64 microseconds
14–12	L0s_EX_LAT	L0s exit latency. 0b101 indicates 1024 ns to less than 2048 ns
11–10	ASPM	Active state power management (ASPM) Support, L0s Entry Supported
9–4	MAX_LINK_W	Maximum link width 0b000001 ×1
3–0	MAX_LINK_SP	Maximum link speed, 0b0001 indicates 2.5 Gb/s

### 14.4.4.13 PCI Express Link Control Register

The PCI Express link control register is shown in Figure 14-53.

Offset 0x05C

Access: Read/Write



**Figure 14-53. PCI Express Link Control Register**

Table 14-51 describes the PCI Express link control register fields.

**Table 14-51. PCI Express Link Control Register Fields Description**

Bits	Name	Description
15–8	—	Reserved
7	EXT_SYNC	Extended synch
6	CCC	Common clock configuration
5	RL	Retrain link
4	LD	Link disable
3	RCB	Read completion boundary
2	—	Reserved
1–0	ASPM_CTL	Active state power management (ASPM) control

### 14.4.4.14 PCI Express Link Status Register

The PCI Express link status register is shown in [Figure 14-54](#).

Offset 0x05E

Access: Read-only

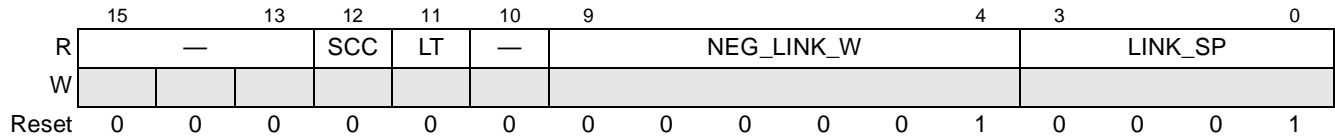


Figure 14-54. PCI Express Link Status Register

[Table 14-52](#) describes the PCI Express link status register fields.

Table 14-52. PCI Express Link Status Register Fields 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

### 14.4.4.15 PCI Express Slot Capabilities Register

The PCI Express slot capabilities register is shown in [Figure 14-55](#). For End Point applications implementing a slot, the content of this register can be modified using the PCI Express Slot Capabilities Update Register as described in [Section 14.4.6.11](#), “PCI Express Slot Capabilities Update Register (PEX\_SLCAP\_UPDATE).”

Offset 0x060

Access: Read-only

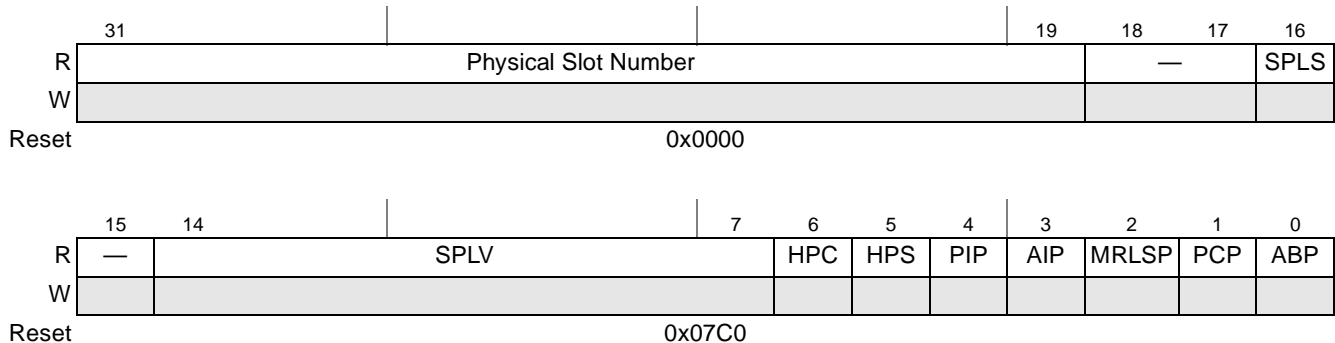


Figure 14-55. PCI Express Slot Capabilities Register

Table 14-53 describes the PCI Express link status register fields.

**Table 14-53. PCI Express Slot Capabilities Register Fields Description**

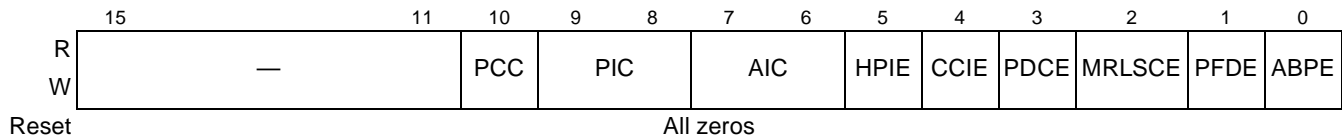
Bits	Name	Description
31–19	Physical Slot Number	This field indicates the physical slot number attached to this Port.
18–17	—	Reserved
16–15	SPLS	Slot power limit scale.
14–17	SPLV	Slot power limit value
6	HPD	Hot plug capable
5	HPS	Hot plug surprise
4	PIP	Power indicator present
3	AIP	Attention indicator present
2	MRLSP	MRL sensor present
1	PCP	Power controller present
0	ABP	Attention button present

#### 14.4.4.16 PCI Express Slot Control Register

The PCI Express slot control register is shown in Figure 14-56.

Offset 0x064

Access: Read/Write



**Figure 14-56. PCI Express Slot Control Register**

Table 14-54 describes the PCI Express slot control register fields.

**Table 14-54. PCI Express Slot Control Register Fields 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



**Table 14-54. PCI Express Slot Control Register Fields Description (continued)**

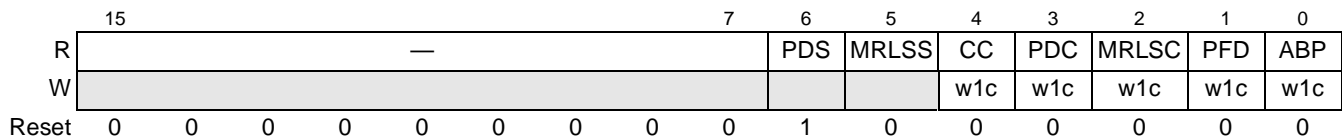
Bits	Name	Description
1	PFDE	Power fault detected enable
0	ABPE	Attention button pressed enable

#### 14.4.4.17 PCI Express Slot Status Register

The PCI Express slot status register is shown in [Figure 14-57](#).

Offset 0x066

Access: Mixed

**Figure 14-57. PCI Express Slot Status Register**

[Table 14-55](#) describes the PCI Express slot status register fields.

**Table 14-55. PCI Express Slot Status Register Fields Description**

Bits	Name	Description
15–7	—	Reserved
6	PDS	Presence detect state. Indicates whether a card is present in the slot. 0 Slot empty 1 Card is present
5	MRLSS	MRL sensor state 0 MRL closed 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

#### 14.4.4.18 PCI Express Root Control Register (RC Mode Only)

The PCI Express root control register is shown in [Figure 14-58](#).

Offset 0x068

Access: Read/Write

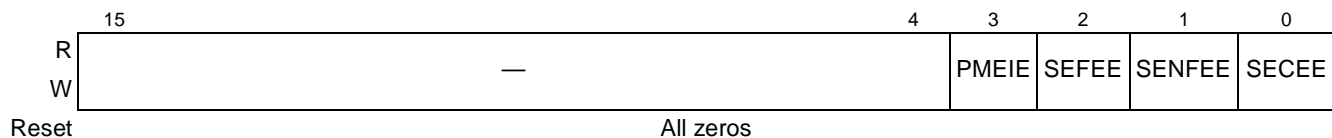
**Figure 14-58. PCI Express Root Control Register**

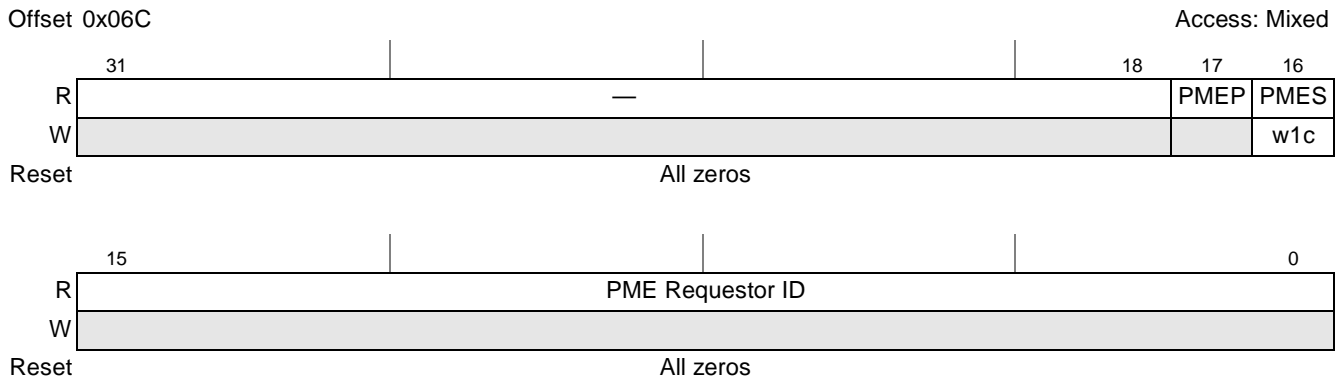
Table 14-56 describes the PCI Express root control register fields.

**Table 14-56. PCI Express Root Control Register Fields 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.

### 14.4.4.19 PCI Express Root Status Register (RC Mode Only)

The PCI Express root status register is shown in Figure 14-59.



**Figure 14-59. PCI Express Root Status Register**

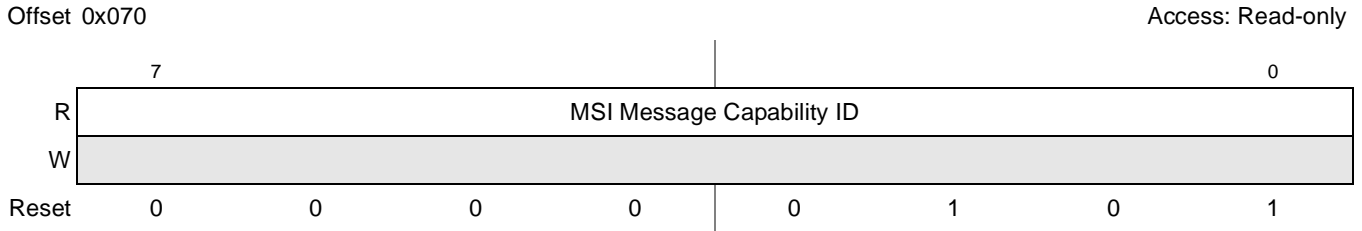
Table 14-57 describes the PCI Express root status register fields.

**Table 14-57. PCI Express Root Status Register Fields Description**

Bits	Name	Description
31–18	—	Reserved
17	PMEP	PME pending.
16	PMES	PME status.
15–0	PME Requestor ID	PME requestor ID.

### 14.4.4.20 PCI Express MSI Message Capability ID Register (EP Mode Only)

The PCI Express MSI message capability ID register is shown in [Figure 14-60](#).



**Figure 14-60. PCI Express Capability ID Register**

[Table 14-58](#) describes the PCI Express capability ID register fields.

**Table 14-58. PCI Express Capability ID Register Fields Description**

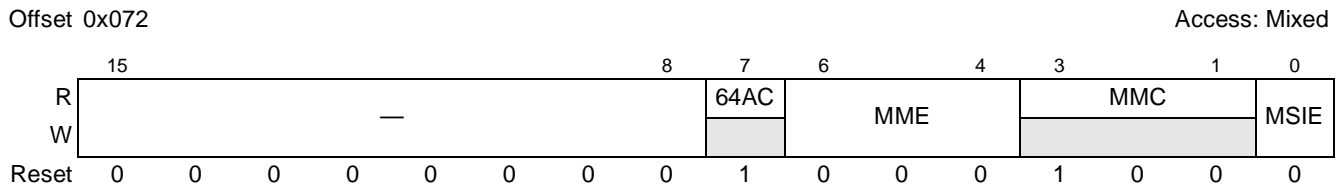
Bits	Name	Description
7-0	MSI Message Capability ID	MSI Message = 0x05

**NOTE**

The value of the Next Pointer register at offset 0x071 is 0x00 (NULL), as this is the last capability of the list.

### 14.4.4.21 PCI Express MSI Message Control Register (EP Mode Only)

The PCI Express MSI message control register is shown in [Figure 14-61](#).



**Figure 14-61. PCI Express MSI Message Control Register**

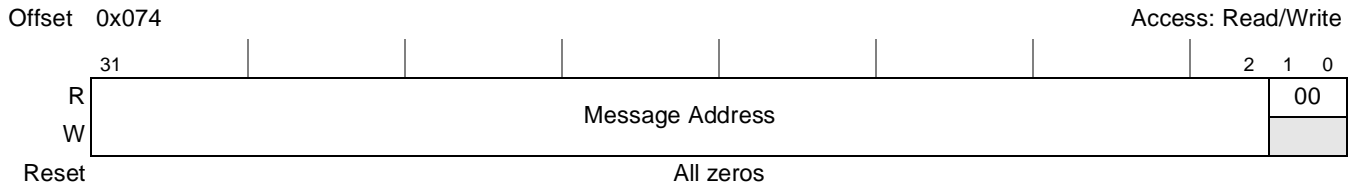
[Table 14-59](#) describes the PCI Express MSI message control register fields.

**Table 14-59. PCI Express MSI Message Control Register Fields 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

### 14.4.4.22 PCI Express MSI Message Address Register (EP Mode Only)

The PCI Express MSI message address register is shown in [Figure 14-62](#).



**Figure 14-62. PCI Express MSI Message Address Register**

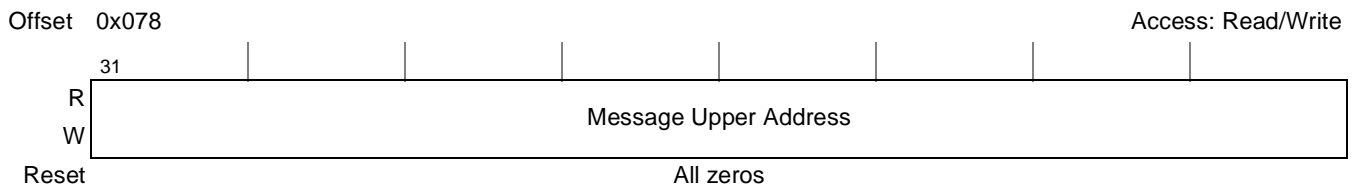
[Table 14-60](#) describes the PCI Express MSI message address register fields.

**Table 14-60. PCI Express MSI Message Address Register Fields 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.

### 14.4.4.23 PCI Express MSI Message Upper Address Register (EP Mode Only)

The PCI Express MSI message upper address register is shown in [Figure 14-63](#).



**Figure 14-63. PCI Express MSI Message Upper Address Register**

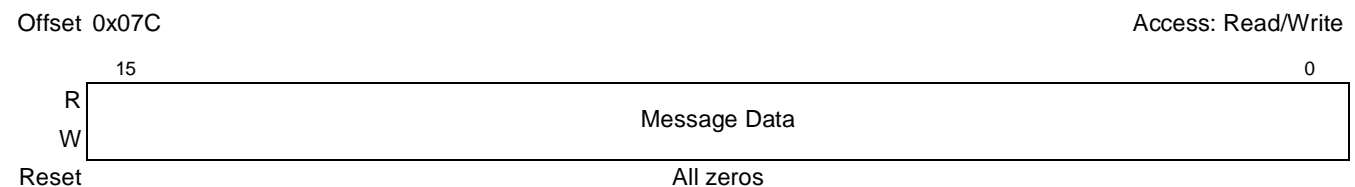
[Table 14-61](#) describes the PCI Express MSI message upper address register fields.

**Table 14-61. PCI Express MSI Message Upper Address Register Fields Description**

Bits	Name	Description
31–0	Message Upper Address	System-specified message upper address

### 14.4.4.24 PCI Express MSI Message Data Register (EP Mode Only)

The PCI Express MSI message data register is shown in [Figure 14-64](#).



**Figure 14-64. PCI Express MSI Message Data Register**

Table 14-62 describes the PCI Express MSI message data register fields.

**Table 14-62. PCI Express MSI Message Data Register Fields Description**

Bits	Name	Description
15–0	Message Data	System-specified message

## 14.4.5 PCI Express Extended Configuration Space

Figure 14-65 shows the PCI Express extended configuration space.

Reserved	Address Offset (Hex)
PCI Express-Compatible Configuration Header (See Section 14.4.1, "Common PCI Express-Compatible Configuration Header Registers," for more information.)	000 03F
PCI Express-Compatible Device-Specific Configuration Space (See Section 14.4.4, "PCI Express-Compatible Device-Specific Configuration Space Registers," for more information.)	040 0FF
Next Capability Offset (NULL <sup>1</sup> )/Capability Version	100
Advanced Error Reporting Capability ID	104
Uncorrectable Error Status	108
Uncorrectable Error Mask	10C
Uncorrectable Error Severity	110
Correctable Error Status	114
Correctable Error Mask	118
Advanced Error Capabilities and Control	11C 120 124 128
Header Log	12C
Root Error Command	130
Root Error Status	134
Error Source ID	138
Correctable Error Source ID	3FF
PCI Express Controller Internal CSRs	400 5A3 5A4 FFF

**Figure 14-65. PCI Express Extended Configuration Space**

<sup>1</sup> Even though the default value of this field is not NULL, it should be considered so by the software.

### 14.4.5.1 PCI Express Advanced Error Reporting Capability ID Register

The PCI Express advanced error reporting capability ID register is shown in Figure 14-66.

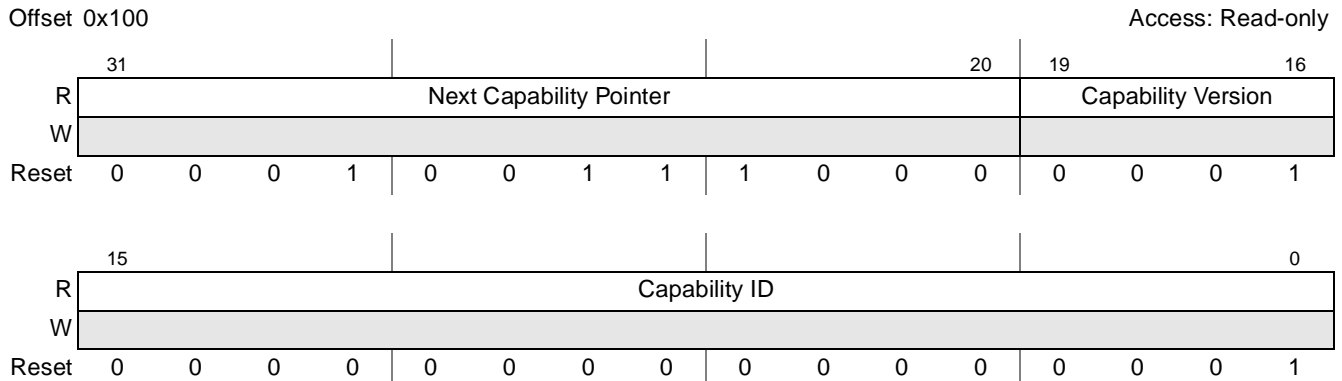


Figure 14-66. PCI Express Advanced Error Reporting Capability ID Register

Table 14-63 describes the PCI Express advanced error reporting capability ID register fields.

Table 14-63. PCI Express Advanced Error Reporting Capability ID Register Fields Description

Bits	Name	Description
13–20	Next Capability Pointer	<b>Note:</b> even though the default value of this field is not NULL, it should be considered so by the software.
19–16	Capability Version	—
15–0	Capability ID	Advanced error reporting capability.

### 14.4.5.2 PCI Express Uncorrectable Error Status Register

The PCI Express uncorrectable error status register is shown in Figure 14-67. When a particular bit of this status register is set, it indicates that the error has occurred.



Figure 14-67. PCI Express Uncorrectable Error Status Register

Downloaded from Elcodis.com electronic components distributor

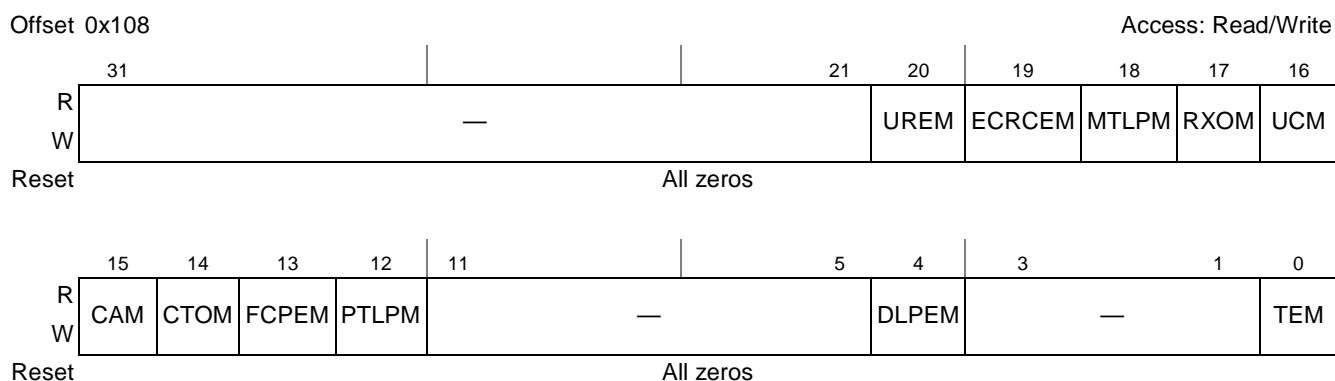
Table 14-64 describes the PCI Express uncorrectable error status register fields.

**Table 14-64. PCI Express Uncorrectable Error Status Register Fields 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
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

### 14.4.5.3 PCI Express Uncorrectable Error Mask Register

The PCI Express uncorrectable error mask register is shown in Figure 14-68. An error is masked if the corresponding mask bit in this register is set.



**Figure 14-68. PCI Express Uncorrectable Error Mask Register**





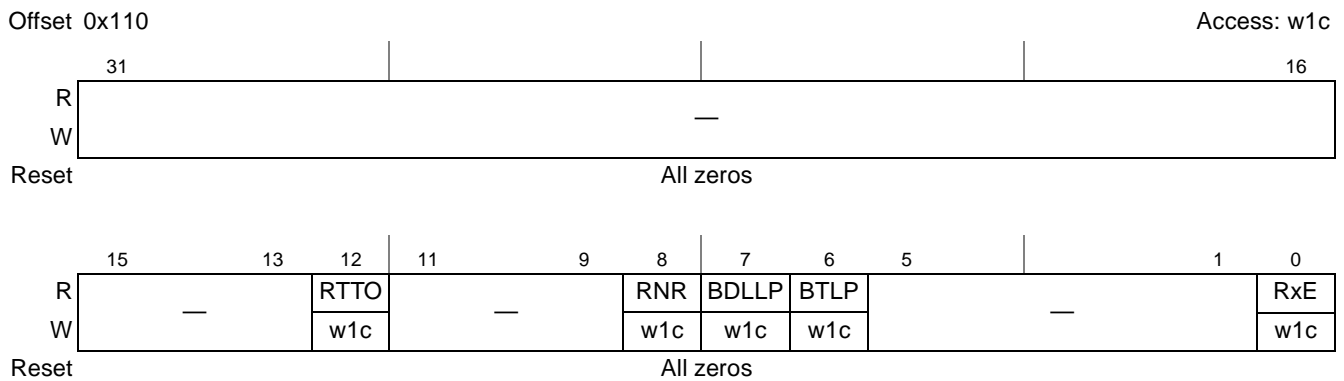
Table 14-66 describes the PCI Express uncorrectable error severity register fields.

**Table 14-66. PCI Express Uncorrectable Error Severity Register Fields 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

### 14.4.5.5 PCI Express Correctable Error Status Register

The PCI Express correctable error status register is shown in Figure 14-70. When an individual error status bit of this read-only register is set, it indicates that this particular error has occurred.



**Figure 14-70. PCI Express Correctable Error Status Register**

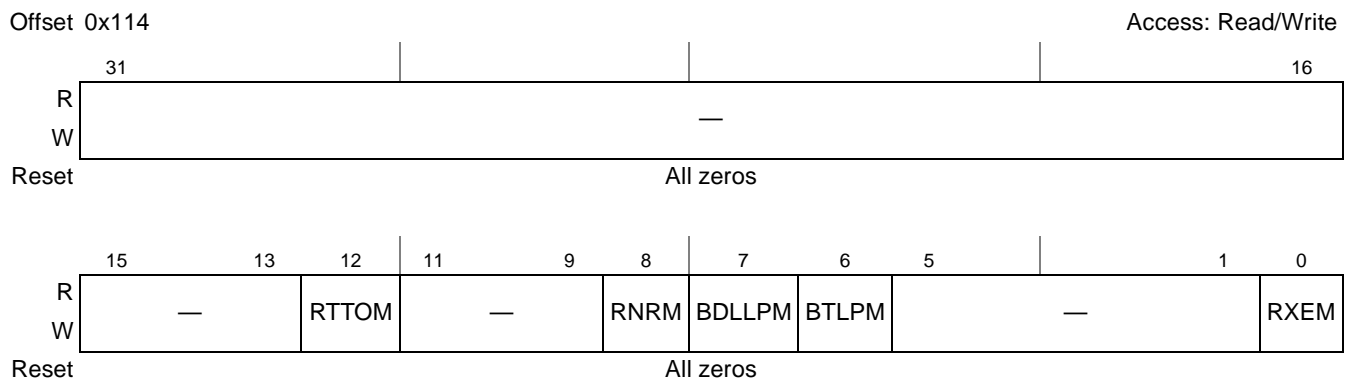
Table 14-67 describes the PCI Express correctable error status register fields.

**Table 14-67. PCI Express Correctable Error Status Register Fields Description**

Bits	Name	Description
31–13	—	Reserved
12	RTTO	Replay timer time-out 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

### 14.4.5.6 PCI Express Correctable Error Mask Register

The PCI Express correctable error mask register is shown in Figure 14-71. An error is masked if the corresponding mask bit in this register is set.



**Figure 14-71. PCI Express Correctable Error Mask Register**

Table 14-68 describes the PCI Express correctable error mask register fields.

**Table 14-68. PCI Express Correctable Error Mask Register Fields Description**

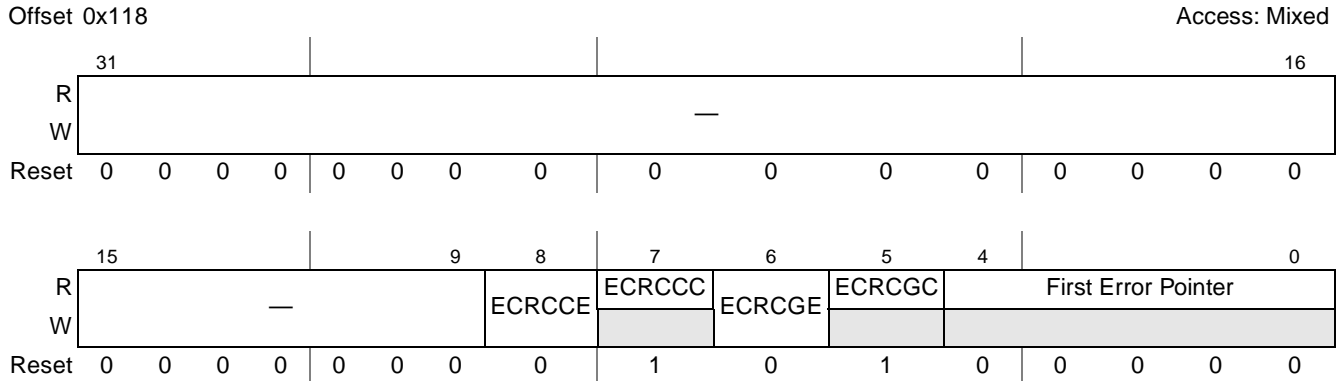
Bits	Name	Description
31–13	—	Reserved
12	RTTOM	Replay timer timeout mask
11–9	—	Reserved
8	RNRM	REPLAY_NUM rollover mask
7	BDLLPM	Bad DLLP mask
6	BTLPM	Bad TLP mask

**Table 14-68. PCI Express Correctable Error Mask Register Fields Description (continued)**

Bits	Name	Description
5–1	—	Reserved
0	RXEM	Receiver error mask

### 14.4.5.7 PCI Express Advanced Error Capabilities and Control Register

The PCI Express advanced error capabilities and control register is shown in [Figure 14-72](#).



**Figure 14-72. PCI Express Advanced Error Capabilities and Control Register**

[Table 14-69](#) describes the PCI Express advanced error capabilities and control register fields.

**Table 14-69. PCI Express Advanced Error Capabilities and Control Register Fields Description**

Bits	Name	Description
31–9	—	Reserved
8	ECRCCE	ECRC checking enable. Set this bit to enable ECRC checking.
7	ECRCCE	ECRC checking capable. Status bit indicates if this capability has been enabled. 0 ECRC checking capability is disabled. 1 ECRC checking capability is enabled.
6	ECRCGE	ECRC generation enable. Set this bit to enable ECRC generation.
5	ECRCGC	ECRC generation capable. Status bit indicates if this capability has been enabled. 0 ECRC generation capability is disabled. 1 ECRC generation capability is enabled.
4–0	First Error Pointer	First error pointer. Identifies the bit position of the first error reported in uncorrectable error status register, <a href="#">Section 14.4.5.2, “PCI Express Uncorrectable Error Status Register.”</a>

### 14.4.5.8 PCI Express Header Log Register

The PCI Express header log register is shown in [Figure 14-73](#).

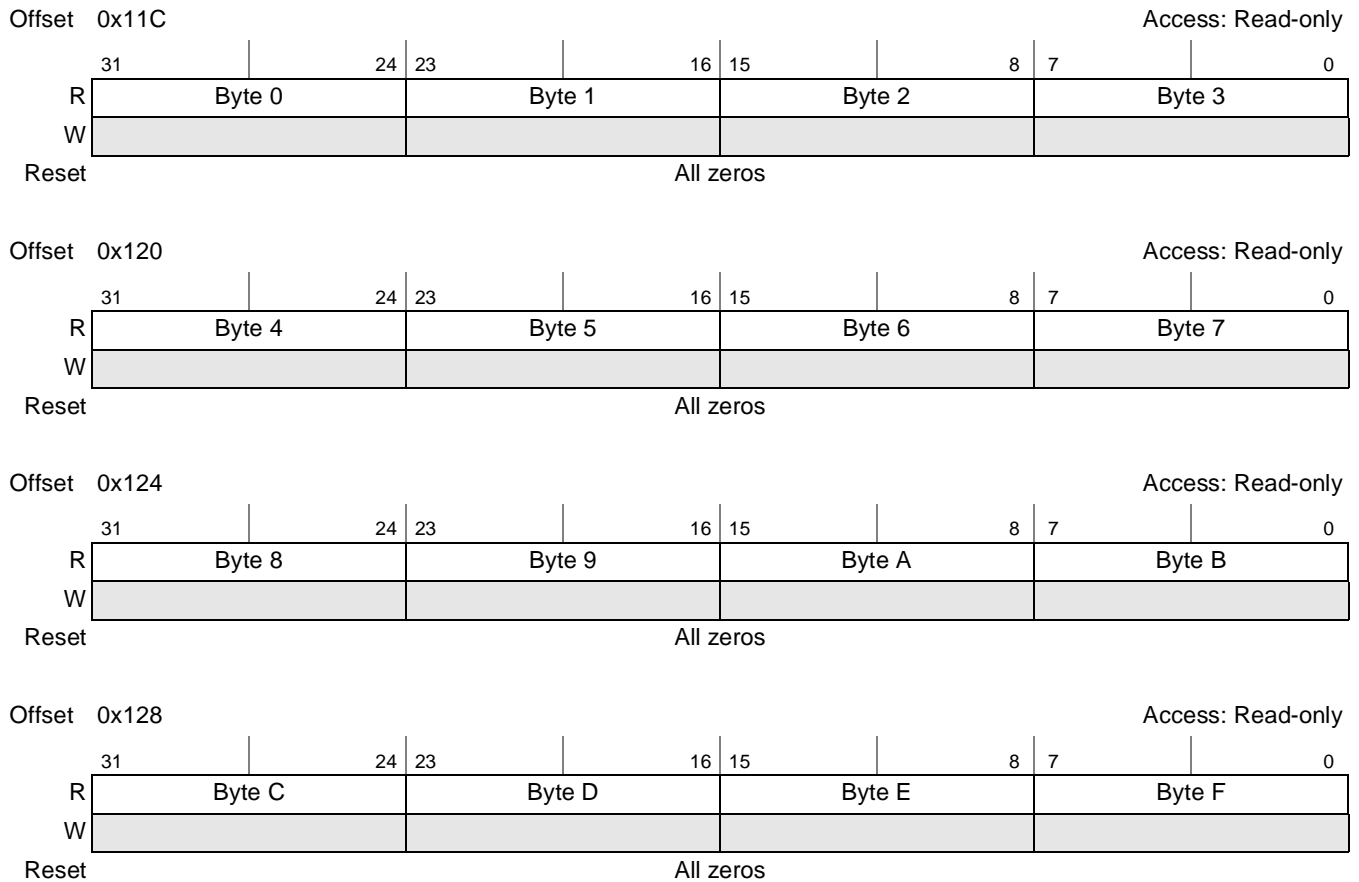


Figure 14-73. PCI Express Header Log Register

Table 14-70 describes the PCI Express header log register fields.

Table 14-70. PCI Express Header Log Register Fields Description

Bits	Name	Description
127–0	Header Log	Header of TLP associated with error.

### 14.4.5.9 PCI Express Root Error Command Register

The PCI Express root error command register is shown in Figure 14-74.

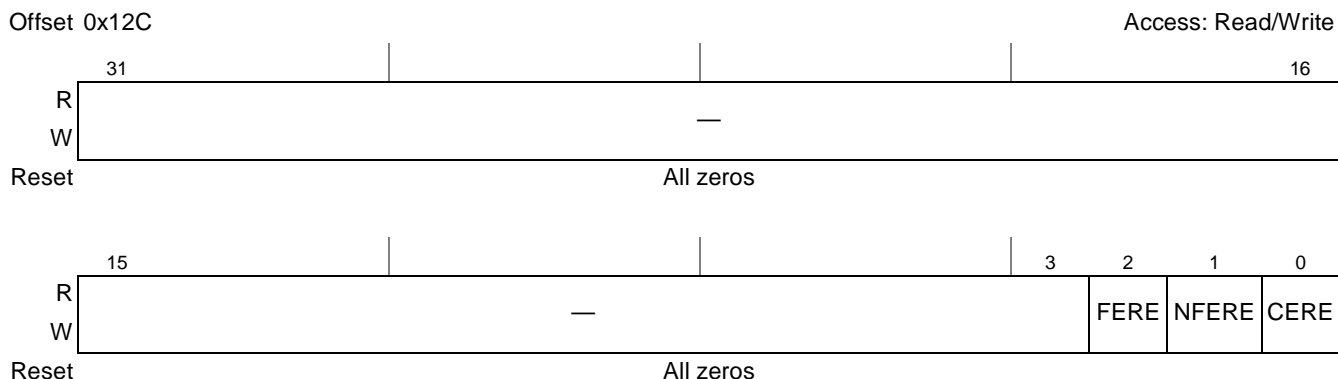


Figure 14-74. PCI Express Root Error Command Register

Table 14-71 describes the PCI Express root error command register fields.

Table 14-71. PCI Express Root Error Command Register Fields 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

### 14.4.5.10 PCI Express Root Error Status Register

The PCI Express root error status register is shown in Figure 14-75.

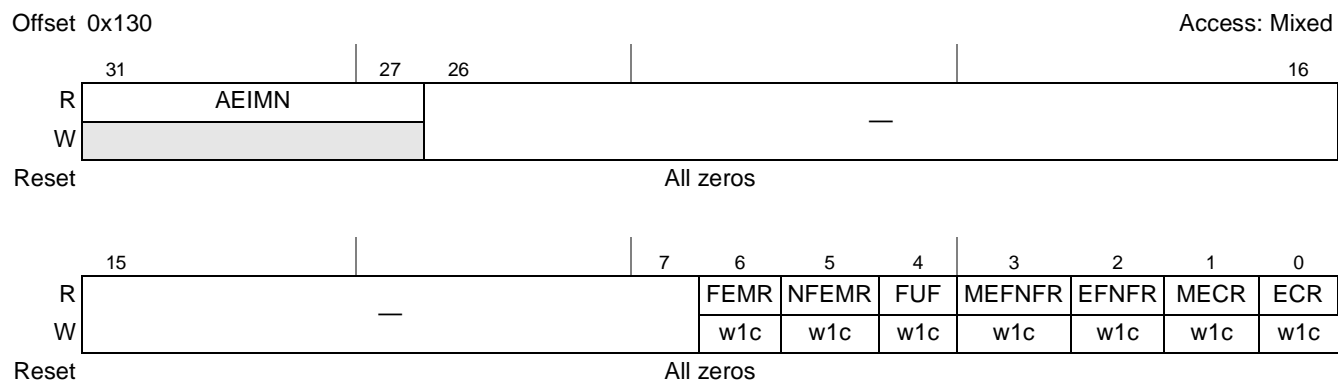


Figure 14-75. PCI Express Root Error Status Register

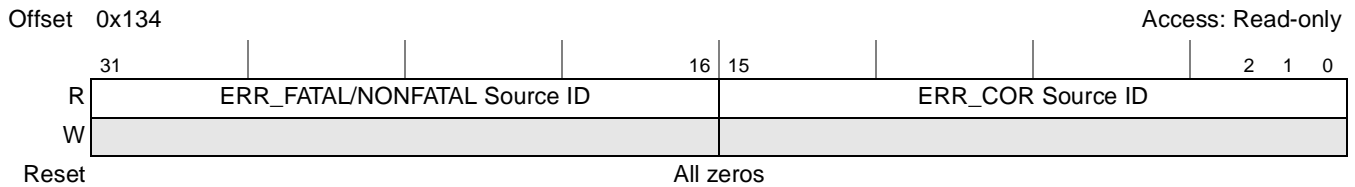
Table 14-72 describes the PCI Express root error command register fields.

**Table 14-72. PCI Express Root Error Status Register Fields Description**

Bits	Name	Description
31–27	AEIMN	Advanced error interrupt message number.
26–7	—	Reserved
6	FEMR	Fatal error messages received.
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.

### 14.4.5.11 PCI Express Error Source Identification Register

The error source identification register shown in Figure 14-76 identifies the source (Requestor ID) of first correctable and uncorrectable (non-fatal/fatal) errors reported in the root error status register. This register is relevant for RC only.



**Figure 14-76. PCI Express Error Source Identification Register**

Table 14-73 describes the PCI Express root source identification register fields.

**Table 14-73. PCI Express Error Source Identification Register Fields Description**

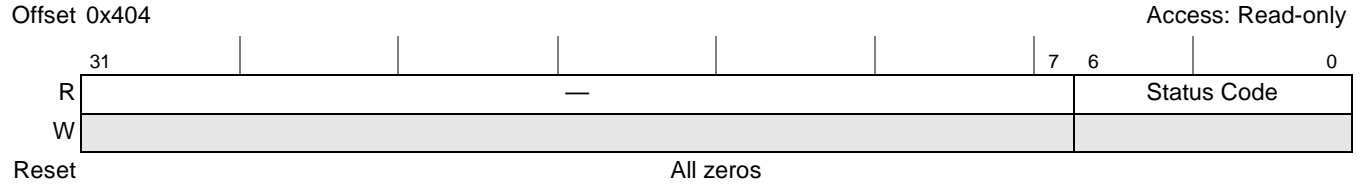
Bits	Name	Description
31–16	ERR_FATAL/NONFATAL Source ID	ERR_FATAL/NONFATAL Source Identification. Loaded with the Requestor ID indicated in the received ERR_FATAL or ERR_NONFATAL Message when the ERR_FATAL/NONFATAL Received register is not already set.
15–0	ERR_COR Source ID	Correctable Error Source Identification. Loaded with the Requestor ID indicated in the received ERR_COR Message when the ERR_COR Received register is not already set.

## 14.4.6 PCI Express Controller Internal Control and Status Registers (CSRs)

This section describes the PCI Express controller internal control and status registers.

### 14.4.6.1 PCI Express LTSSM State Status Register (PEX\_LTSSM\_STAT)

PEX\_LTSSM\_STAT, shown in [Figure 14-77](#), provides details on link training status. This register is useful for debugging link training failures.



**Figure 14-77. PCI Express LTSSM State Status Register (PEX\_LTSSM\_STAT)**

The fields of the PEX\_LTSSM\_STAT are described in [Table 14-74](#).

**Table 14-74. PEX\_LTSSM\_STAT Fields Description**

Bits	Name	Description
31–7	—	Reserved
6–0	Status code	Status code. See <a href="#">Table 14-75</a> for encodings.

[Table 14-75](#) provides the encodings for the status code field of the PEX\_LTSSM\_STAT register.

**Table 14-75. 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)



Table 14-75. PEX\_LTSSM\_STAT Status Codes (continued)

Status Code (Hex)	LTSSM State Description	Status Code (Hex)	LTSSM State Description
0F	Configuration lane number wait (2)	36	Recovery cfg (1)
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

#### 14.4.6.2 PCI Express N\_FTS Control Register (PEX\_NFTS\_CTRL)

The PCI Express N\_FTS Control Register, shown in [Figure 14-78](#), is used to set the N\_FTS value that is advertised by the PCI Express controller during link training. It is preferable to set it before the PCI Express core internal reset is negated. If this value is changed after the link is up, the new value will take effect during the next link training. The N\_FTS value is should be set according to the L0s exit latency of

the Rx link of PHY. At a given time, either N\_FTS or N\_FTS\_COM value is used based on the setting of common clock configuration bit in the configuration space.

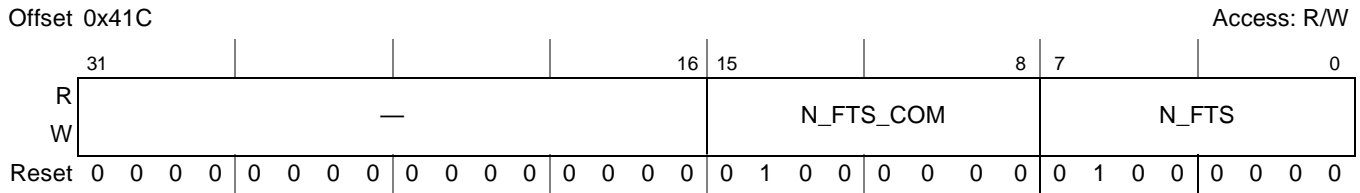


Figure 14-78. PCI Express N\_FTS Control Register

The fields of the PCI Express N\_FTS Control Register are described in Table 14-76.

Table 14-76. PCI Express N\_FTS Control Register Fields Description

Bits	Name	Description
31–16	—	Reserved
15–8	N_FTS_COM	Number of fast training sequence ordered sets in common clock mode. This is the number of Fast training sequence (FTS) ordered sets that the PHY requires to enable its Rx circuits to achieve bit and symbol lock and come out of ASPM L0s link power state when devices on either side of the link use common reference clock. This N_FTS value is advertised by the PCI Express controller to the remote device during link training if the common clock configuration bit in configuration space is set.
7–0	N_FTS	Number of Fast Training Sequence ordered sets. This is the number of fast training sequence (FTS) ordered sets that PHY requires to enable its Rx circuits to achieve bit and symbol lock and come out of ASPM L0s link power state. This N_FTS value is advertised by the PCI Express controller to the remote device during link training. Its value has to be calculated based on the L0s_exit latency time required by the PHY layer circuits.

### 14.4.6.3 PCI Express ACK Replay Timeout Register (PEX\_ACKRPLY\_TO)

The PCI Express ACK Replay Timeout Register, shown in Figure 14-79, is used to program timeout values for ACK DLLP transmission and reception in the DLL. Ack receive timeout is termed as Replay timeout since TLPs are retransmitted after this timeout. Both values should be in terms of system clock cycles number and have to be set based on Max-Payload-size and operating link width as specified by the protocol.

Ack time-out will also depend upon ASPM L0s enabling for the Tx link of the device. The PCI Express controller implements a look-up table for automatic updates of the ack and replay time-out values, based on max-payload size, link\_width and ASPM L0s enabled information. There for the reset value of this register may be different that shown in the figure. The automatic updating of these values will be disabled upon the first write to this register assuming that the software will take care of the updates based on the above factors from then on.

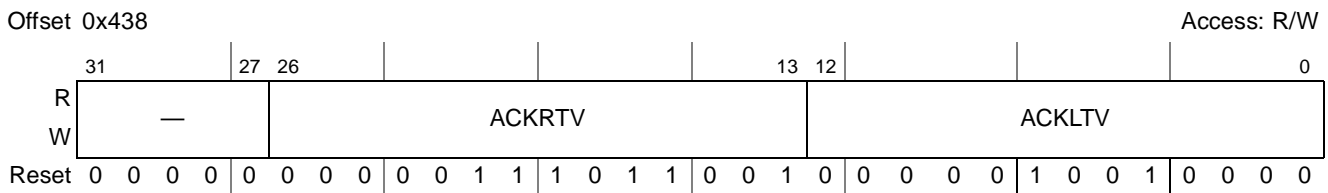


Figure 14-79. PCI Express ACK Replay Timeout Register

Table 14-77 describes PCI Express N\_FTS control register fields.

**Table 14-77. PCI Express ACK Replay Timeout Register Fields Description**

Bits	Name	Description
31–27	—	Reserved
26–13	ACKRTV	<p>Ack Replay Timeout Value. Timeout value to wait for reception of ACK DLLP from the link side by the DLL before re-transmitting TLPs. The protocol specifies this value in symbol times for various combinations of max-payload size and negotiated link width. The value programmed into this field should be in terms of system clock cycles number, and can be calculated as:</p> $\frac{(\text{REPLAY\_TIMER\_TIMEOUT} + \text{Rx\_L0s\_Adjustment}) \times \text{SYSTEM\_CLOCK}}{250}$ <ul style="list-style-type: none"> <li>REPLAY_TIMER_TIMEOUT—Timeout value for the replay timer, specified in symbol times.</li> <li>Rx_L0s_Adjustment—The time required by the component's receive circuits to exit from L0s to L0 specified in symbol times.</li> <li>SYSTEM_CLOCK—The PCI Express controller system clock, specified in MHz.</li> </ul> <p><b>Note:</b> The “250” denominator represent the frequency of a symbol (250 MHz).</p>
12–0	ACKLTV	<p>Ack Latency Timeout Value. Timeout value to force transmission of ACK DLLP by the DLL after a TLP is received. The protocol specifies this value in symbol times for various combinations of max-payload size &amp; negotiated link width. The value programmed into this field should be in terms of system clock cycles number, and can be calculated as:</p> $\frac{(\text{ACK\_LATENCY\_TIEMOUT} + \text{Tx\_L0s\_Adjustment}) \times \text{SYSTEM\_CLOCK}}{250}$ <ul style="list-style-type: none"> <li>ACK_LATENCY_TIMEOUT—Timeout value to force transmission of ACK DLLP, specified in symbol times.</li> <li>Tx_L0s_Adjustment—The time required for the Transmitter to exit L0s, specified in symbol times.</li> <li>SYSTEM_CLOCK—The PCI Express controller system clock, specified in MHz.</li> </ul> <p><b>Note:</b> The “250” denominator represent the frequency of a symbol (250 MHz).</p>

#### 14.4.6.4 PCI Express Controller Core Clock Ratio Register (PEX\_GCLK\_RATIO)

The PCI Express controller core clock ratio register, shown in Figure 14-80, is used to program the ratio of the actual PCI Express controller core clock (csb\_clk divided according to SCCR[PCIEXPnCM]) frequency to the maximum possible controller core frequency (125 MHz). Changing the default value of this register is required only when a PCI Express controller core clock frequency is different than its maximum.

This ratio will be used by the PCI Express controller only to calculate the actual timer values to be used for Ack Latency and Replay timeout values. These two timer values have to dynamically change based on the negotiated link-width and max-payload size. The default value by itself is not enough for these two timers. By programming the clock ratio in this register, the calculation is automatically adjusted by hardware. Note that other timer registers in the PCI Express controller may still have to be programmed to a new value based on the actual controller core clock used in the specific application.

#### NOTE

The default PCI Express controller core clock is csb\_clk (SCCR[PCIEXPnCM] = 0b01).





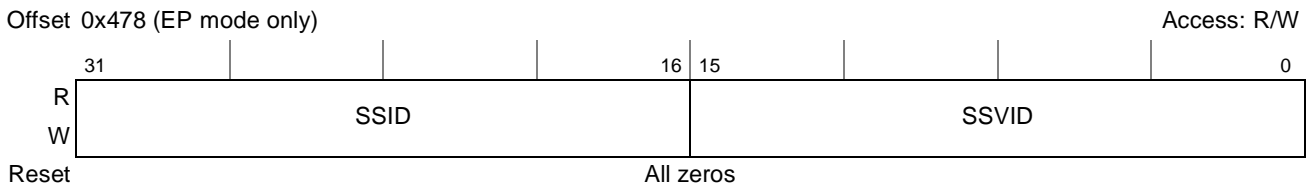
Table 14-81 describes the PCI Express ASPM request timer register fields.

**Table 14-81. PCI Express ASPM Request Timer Register Fields Description**

Bits	Name	Description
31–13	—	Reserved
12–0	ASPML1TMR	ASPM L1 request timer value. This is the time-out interval after sending NAK message, before a new ASPM L1 request from a downstream device is treated as a new ASPM L1 entry request. For example, if the upstream device rejects an ASPM L1 entry request from a downstream device with ASPM NAK message, the next ASPM L1 entry request from downstream device will be entertained only after this timeout interval or only after the Rx link of the downstream port enters L0s state. This value is specified in terms of system clock cycles (CSB clock/PCIEXPnCM). This value can be calculated as [Time in microseconds × SYSTEM_CLK in MHz]. For example 9.5[μs] × 125[MHz] = 1187 (0x4A3). This register is used only in RC mode.

#### 14.4.6.8 PCI Express Subsystem Vendor ID Update Register (PEX\_SSVID\_UPDATE)

The PCI Express subsystem vendor ID update register (PEX\_SSVID\_UPDATE) shown in Figure 14-84 is used to configure the subsystem vendor ID and subsystem ID fields of the configuration header (offset 0x2C) for End Point devices. This register has to be programmed before setting the config-ready bit in the PCI express configuration ready register so that the host reads the correct information during enumeration.



**Figure 14-84. PCI Express Subsystem Vendor ID Update Register (PEX\_SSVID\_UPDATE)**

**Table 14-82. PEX\_SSVID\_UPDATE Fields Description**

Bits	Name	Description
31–16	SSID	Subsystem ID
15–0	SSVID	Subsystem Vendor ID

#### 14.4.6.9 PCI Express Device Capabilities Update Register (PEX\_DEVCAP\_UPDATE)

The PCI Express device capabilities update register shown in Figure 14-85 is used to set the values to the PCI Express device capabilities register in the PCI Express configuration header (offset 0x50). It can be used when the device is configured as an End Point to make the correct device information available to the upstream device. This register has to be programmed before setting the config-ready bit in the PCI Express configuration ready register so that the host reads the correct information during enumeration.

Offset 0x47C

Access: R/W

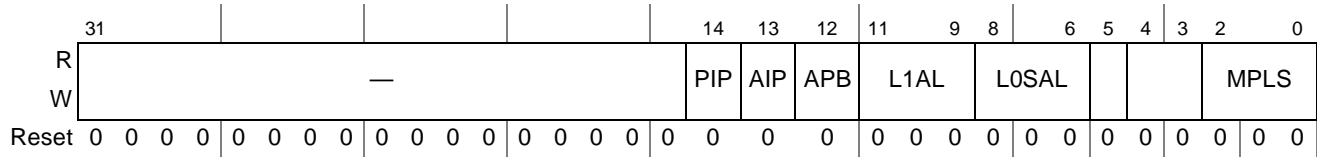


Figure 14-85. PCI Express Device Capabilities Update Register

The fields of the PCI Express Device Capabilities Update Register are described in [Table 14-83](#).

Table 14-83. PCI Express Device Capabilities Update Register Fields Description

Bits	Name	Description
31–15	—	Reserved
14	PIP	Power Indicator Present
13	AIP	Attention Indicator Present
12	APB	Attention Button Present
11–9	L1AL	Endpoint L1 Acceptable Latency. This field indicates the acceptable latency that an Endpoint can withstand due to the transition from L1 state to the L0 state. Defined encodings are: 000b Less than 1 $\mu$ s 001b 1 $\mu$ s to less than 2 $\mu$ s 010b 2 $\mu$ s to less than 4 $\mu$ s 011b 4 $\mu$ s to less than 8 $\mu$ s 100b 8 $\mu$ s to less than 16 $\mu$ s 101b 16 $\mu$ s to less than 32 $\mu$ s 110b 32 $\mu$ s-64 $\mu$ s 111b More than 64 $\mu$ s
8–6	L0SAL	Endpoint L0s Acceptable Latency. This field indicates the acceptable total latency that an Endpoint can withstand due to the transition from L0s state to the L0 state. Defined encodings are: 000b Less than 64 ns 001b 64 ns to less than 128 ns 010b 128 ns to less than 256 ns 011b 256 ns to less than 512 ns 100b 512 ns to less than 1 $\mu$ s 101b 1 $\mu$ s to less than 2 $\mu$ s 110b 2 $\mu$ s-4 $\mu$ s 111b More than 4 $\mu$ s
5	—	Reserved (Extended Tag Field Supported). Must be set to 0b.
4–3	—	Reserved (Phantom Functions Supported). Must be set to 00b.
2–0	MPLS	Max Payload Size Supported. Must be set to 000b (128bytes)

#### 14.4.6.10 PCI Express Link Capabilities Update Register (PEX\_LINKCAP\_UPDATE)

The PCI Express link capabilities update register shown in [Figure 14-86](#) is used to set the values to the PCI Express link capabilities register in the PCI Express configuration header (offset 0x58). It can be used when the device is configured as an End Point to make the correct link information available to the

upstream device. This register has to be programmed before setting the config-ready bit in the PCI Express configuration ready register so that the host reads the correct information during enumeration.

Offset 0x480

Access: R/W

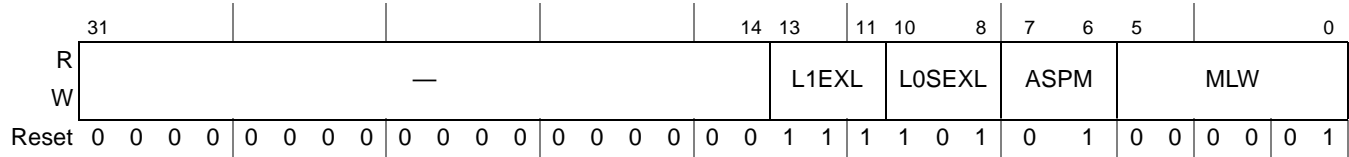


Figure 14-86. PCI Express Link Capabilities Update Register

The fields of the PCI Express link capabilities update register are described in Table 14-84.

Table 14-84. PCI Express Link Capabilities Update Register Fields Description

Bits	Name	Description
31–14	—	Reserved
13–11	L1EXL	L1 Exit Latency for the given PCI Express Link. The value reported indicates the length of time this port requires to complete transition from L1 to L0. Defined encodings are: 000b Less than 1 $\mu$ s 001b 1 $\mu$ s to less than 2 $\mu$ s 010b 2 $\mu$ s to less than 4 $\mu$ s 011b 4 $\mu$ s to less than 8 $\mu$ s 100b 8 $\mu$ s to less than 16 $\mu$ s 101b 16 $\mu$ s to less than 32 $\mu$ s 110b 32 $\mu$ s to 64 $\mu$ s 111b More than 64 $\mu$ s <b>Note:</b> Exit latencies may be influenced by PCI-Express reference clock configuration depending upon whether a component uses a common or separate reference clock.
10–8	LOSEXL	L0s Exit Latency for the given PCI Express Link. The value reported indicates the length of time this port requires to complete transition from L0s to L0. Defined encodings are: 000b Less than 64 ns 001b 64 ns to less than 128 ns 010b 128 ns to less than 256 ns 011b 256 ns to less than 512 ns 100b 512 ns to less than 1 $\mu$ s 101b 1 $\mu$ s to less than 2 $\mu$ s 110b 2 $\mu$ s to 4 $\mu$ s 111b More than 4 $\mu$ s <b>Note:</b> Exit latencies may be influenced by PCI Express reference clock configuration depending upon whether a component uses a common or separate reference clock. This field is automatically updated to a new value if the common clock configuration bit in config space is set by host.
7–6	ASPM	ASPM Support. Indicates the level of ASPM supported on the given PCI Express Link. Defined encodings are: 00b Reserved 01b L0s Entry Supported 10b Reserved 11b Reserved (L0s and L1 not supported by this device)
5–0	MLW	Maximum Link Width of the given PCI Express Link. Defined encodings are: 000001b $\times$ 1 Other: Reserved



### 14.4.6.11 PCI Express Slot Capabilities Update Register (PEX\_SLCAP\_UPDATE)

The PCI Express slot capabilities update register shown in Figure 14-87 is used to set the values to the PCI Express slot capabilities register in the PCI Express configuration header (offset 0x60). It can be used when the device is configured as an End Point to make the correct slot information available to the upstream device. This register has to be programmed before setting the config-ready bit in the PCI Express configuration ready register so that the host reads the correct information during enumeration.

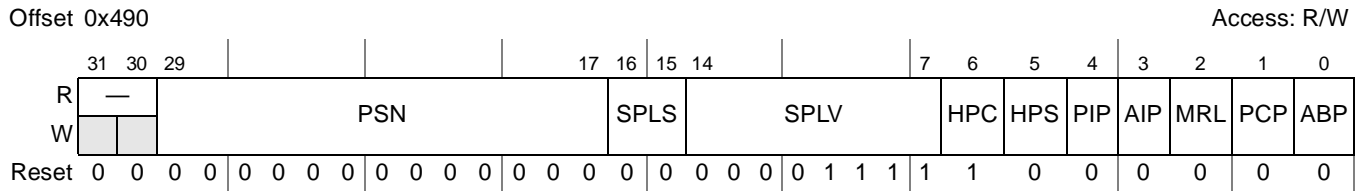


Figure 14-87. PCI Express Slot Capabilities Update Register

Table 14-85 shows the PCI Express slot capabilities update register fields description.

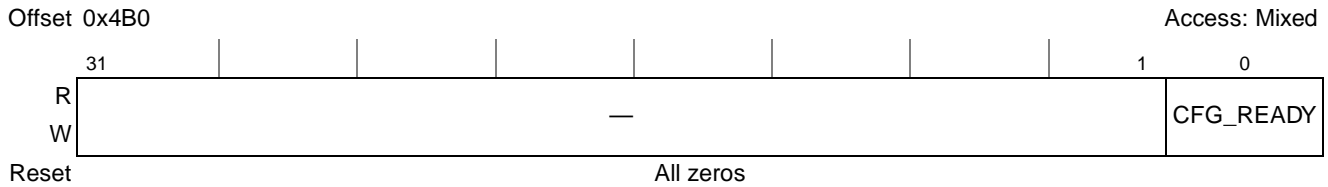
Table 14-85. PCI Express Slot Capabilities Update Register Fields Description

Bits	Name	Description
31–30	—	Reserved
29–17	PSN	Physical Slot Number. Physical slot number attached to the port.
16–15	SPLS	Slot Power Limit Scale. Specifies the scale used for the slot power limit value. The range of values is as follows: 00 1.0x 01 0.1x 10 0.01x 11 0.001x
14–7	SPLV	Slot power limit value. In combination with the slot power limit scale value, specifies the upper limit on power supplied by slot. The power limit (in Watts) is calculated by multiplying the value in this field by the value in the slot power limit scale field.
6	HPC	Hot plug capable
5	HPS	Hot plug surprise
4	PIP	Power indicator present
3	AIP	Attention indicator present
2	MRL	MRL sensor present
1	PCP	Power controller present
0	ABP	Attention button present

### 14.4.6.12 PCI Express Configuration Ready Register

The PCI Express configuration ready register, shown in Figure 14-88, indicates 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). To ensure that the

external host reads the correct capability advertisements during enumeration, the CFG\_READY bit in this register should be set only after all relevant configuration registers are programmed.



**Figure 14-88. PCI Express Configuration Ready Register (PEX\_CFG\_READY)**

The fields of the PCI Express configuration ready register are described in [Table 14-86](#).

**Table 14-86. PEX\_CFG\_READY Fields Description**

Bits	Name	Description
31–1	—	Reserved
0	CFG_READY	Configuration ready 1 The transaction layer accepts inbound configuration requests. 0 The transaction layer responds to all inbound configuration requests with retry (CRS)

### 14.4.7 PCI Express BAR Configuration Registers (EP Mode)

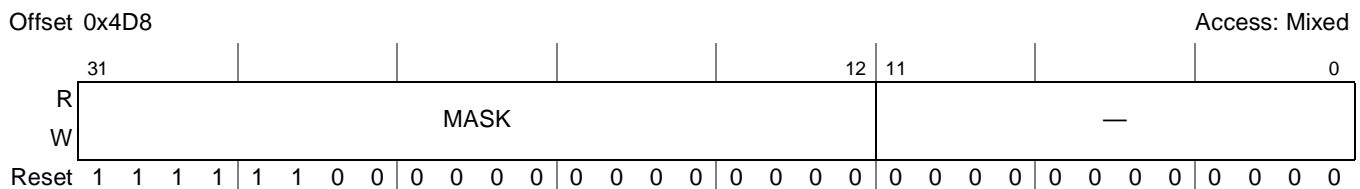
The registers described in this section configure the size and the prefetchable attributes of the base address registers (BAR) in the PCI Express configuration space. The local host should program these attributes before setting the config-ready bit in the PCI Express configuration space so that the RC host reads the correct information during enumeration. These registers are used only in EP mode.

#### 14.4.7.1 PCI Express BAR Size Low Configuration Register (PEX\_BAR\_SIZEL)

PEX\_BAR\_SIZEL, shown in [Figure 14-89](#), configures the size of the 32-bit address windows and low portion of the 64-bit address windows by setting the mask value for the base address registers (BAR) in the PCI Express configuration space. The specific PCI Express BAR is selected by the PCI Express BAR select configuration register (PEX\_BAR\_SEL). Before programming this register, the PEX\_BAR\_SEL register should be programmed to select the correct BAR, whose size needs to be set. This register is used only in EP mode.

#### NOTE

Because the CSB address space of the device is limited to 4 Gigabytes, the high portion of the 64-bit address window mask value is always set to all ones (0xFFFF\_FFFF) to comply with the standard enumeration sequence.



**Figure 14-89. PCI Express BAR Size Low Configuration Register (PEX\_BAR\_SIZEL)**

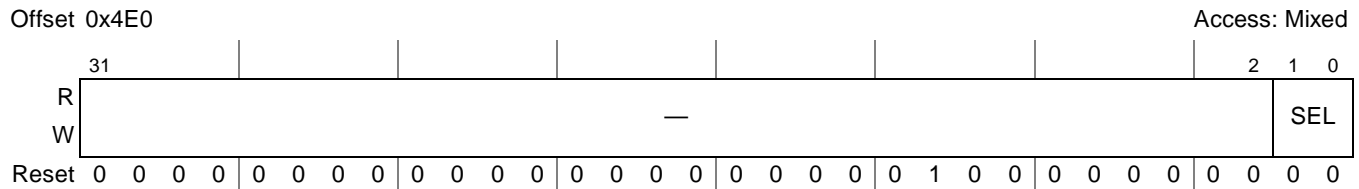
The fields of the PEX\_BAR\_SIZEL register are described in [Table 14-87](#).

**Table 14-87. PEX\_BAR\_SIZEL Fields Description**

Bits	Name	Description
31–12	MASK	Mask. Sets the mask for the BAR, and any bit with a value of zero is masked. When the RC does a configuration write to the BAR during the enumeration sequence, bits that are masked cannot be modified and remain zeros. All ones and zeros in this register must be consecutive. The actual size is according to the location of the least significant bit in the MASK[31–12] field, which is set. If MASK[31–m] is all ones, the size is 2 <sup>m</sup> bytes. If MASK[31–12] is all zeros, the window size is 4 Gigabytes. For example: 1111...1111 - 2 <sup>12</sup> , 4 Kilobytes window. 1111...1110 - 2 <sup>13</sup> , 8 Kilobytes window. ... 1100...0000 - 2 <sup>30</sup> , 1 Gigabytes window. 1000...0000 - 2 <sup>31</sup> , 2Gigabytes window. 0000...0000 - 4 Gigabytes window.
11–0	—	Reserved. Must be zeros

### 14.4.7.2 PCI Express BAR Select Configuration Register (PEX\_BAR\_SEL)

PEX\_BAR\_SEL, shown in [Figure 14-90](#), is used to select the specific BAR for which the size is being configured by the PEX\_BAR\_SIZEL register. This register should be programmed before the PEX\_BAR\_SIZEL register is accessed, and it is used only in EP mode.



**Figure 14-90. PCI Express BAR Select Configuration Register (PEX\_BAR\_SEL)**

The fields of the PEX\_BAR\_SEL register are described in [Table 14-88](#).

**Table 14-88. PEX\_BAR\_SEL Fields Description**

Bits	Name	Description
31–2	—	Reserved. Must be zeros.
1–0	SEL	Select. Selects the specific BAR size to be programmed by the PEX_BAR_SIZEL and PEX_BAR_SIZEH registers. 00 PEX_BAR_SIZEL points to BAR0 in address 0x010 (Window 0, 32 bit address). 01 PEX_BAR_SIZEL points to BAR1 in address 0x014 (Window 1, 32 bit address). 10 PEX_BAR_SIZEL points to BAR2 in address 0x018 (low portion of window 2, 64 bit address). 11 PEX_BAR_SIZEL points to BAR4 in address 0x020 (low portion of window 3, 64 bit address).

### 14.4.7.3 PCI Express BAR Prefetch Configuration Register (PEX\_BAR\_PF)

PEX\_BAR\_PF, shown in [Figure 14-91](#), sets the Prefetchable field in the base address registers (BAR) of the PCI Express configuration space. The local host should program this register before setting the

config-ready bit in the PCI Express configuration space so that the RC host reads the correct information during enumeration. It is used only in EP mode.

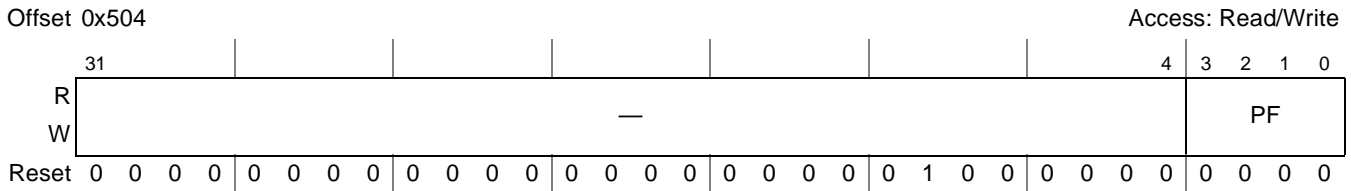


Figure 14-91. PCI Express BAR Prefetch Configuration Register (PEX\_BAR\_PF)

The fields of the PEX\_BAR\_PF register are described in Table 14-89.

Table 14-89. PEX\_BAR\_PF Fields Description

Bits	Name	Description
31–4	—	Reserved. Must be zeros.
3–0	PF	Prefetchable. Sets the prefetchable attribute for all the BARs in the PCI Express configuration space. Each bit determines the prefetchable attribute of the corresponding BAR. PF[3] Prefetchable attribute of BAR4 in address 0x020 (low portion of window 3, 64 bit address) PF[2] Prefetchable attribute of BAR2 in address 0x018 (low portion of window 2, 64 bit address) PF[1] Prefetchable attribute of BAR1 in address 0x014 (Window 1, 32 bit address) PF[0] Prefetchable attribute of BAR0 in address 0x010 (Window 0, 32 bit address)

## 14.4.8 PCI Express Extended Status and Control Registers

### 14.4.8.1 PME\_To\_Ack Timeout Register (RC Mode Only)

PEX\_PME\_TO\_ACK\_TOR, shown in Figure 14-92, is used to program the timeout value for a PME\_To\_Ack message response in terms of PCI Express controller core clock cycles.

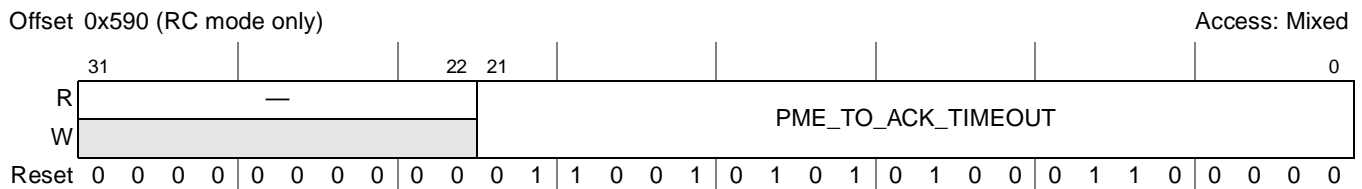


Figure 14-92. PCI Express PME\_To\_Ack Timeout Register (PEX\_PME\_TO\_ACK\_TOR)

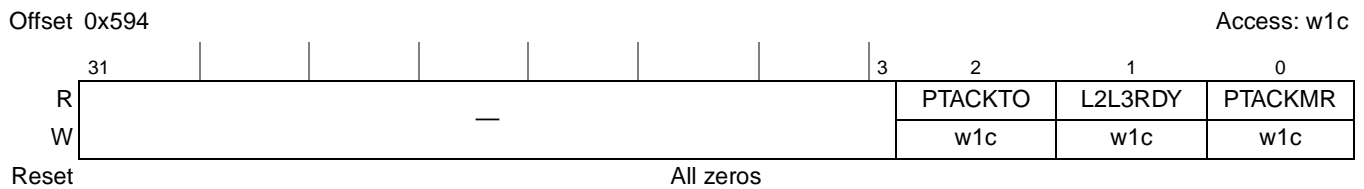
The fields of the PEX\_PME\_TO\_ACK\_TOR are described in [Table 14-90](#).

**Table 14-90. PEX\_PME\_TO\_ACK\_TOR Fields Description**

Bits	Name	Description
31–22	—	Reserved
21–0	PME_TO_ACK_TIMEOUT	After the RC broadcasts a PME_Turn_Off message, 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 because a timeout has occurred. The value is calculated as: Time (in $\mu$ sec) $\times$ PCI Express controller core clock frequency (in MHz) 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.

#### 14.4.8.2 PME\_To\_Ack Status Register (RC Mode Only)

The PME\_To\_Ack Status Register (PEX\_PME\_TO\_ACK\_SR) shown in [Figure 14-93](#), can be used by the power manager software to decide whether it is safe to switch off power to downstream devices, after PME\_Turn\_Off message has been broadcast by the root port.



**Figure 14-93. PME\_To\_Ack Status Register (PEX\_PME\_TO\_ACK\_SR)**

The fields of the PEX\_PME\_TO\_ACK\_SR are described in [Table 14-91](#).

**Table 14-91. PEX\_PME\_TO\_ACK\_SR Fields Description**

Bits	Name	Description
31–3	—	Reserved
2	PTACKTO	PME_To_Ack timeout occurred. This bit is set by the hardware when PME_To_Ack message is not received by the Root Port from downstream device within the timeout duration indicated by PME_To_Ack_timeout register. When this bit is set, it is safe for the Power manager to switch off the power of the downstream device. Once set, this bit will remain set, till software clears it by writing 1'b1 to this bit.
1	L2L3RDY	Entered L2/L3 ready state. This bit is set by hardware when the current power management state is L2/L3 Ready. 100nsec after this bit is set, it is safe for the Power manager to switch off the power of the downstream device. Once set, this bit will remain set, till software clears it by writing 1'b1 to this bit.
0	PTACKMR	PME_To_Ack message received. This bit is set by hardware when PME_To_Ack message is received by the Root Port from the downstream device. When this bit is set, it is safe for the Power manager to switch off the power of the downstream device. Once set, this bit will remain set, till software clears it by writing 1'b1 to this bit.



- Read DMA engine control
- Mailbox
- Message signaled interrupts (MSI) generation
- Events monitoring

**NOTE**

The registers described in this section use little-endian byte ordering. Software running on the local processor in big-endian mode must byte-swap the data. No byte swapping occurs when the registers are accessed from the PCI bus.

### 14.5.2 Global Registers

This section describes the following:

- Section 14.5.2.1, “PCI Express CSB Bridge Control Register (PEX\_CSB\_CTRL)”
- Section 14.5.2.2, “PCI Express DMA Descriptor Timer Register (PEX\_DMA\_DSTMR)”
- Section 14.5.2.3, “PCI Express CSB Bridge Status Register (PEX\_CSB\_STAT)”

#### 14.5.2.1 PCI Express CSB Bridge Control Register (PEX\_CSB\_CTRL)

PEX\_CSB\_CTRL, shown in Figure 14-95, controls the operation of the PCI Express to CSB bridge.

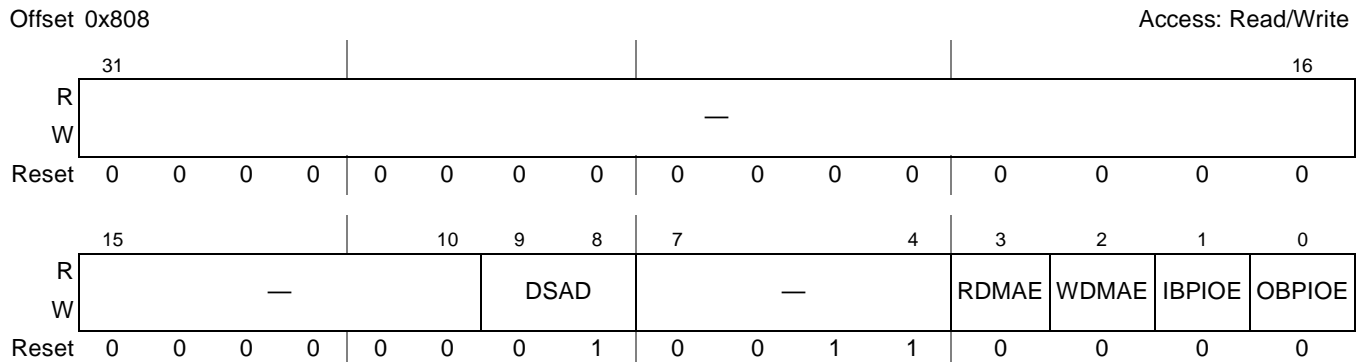


Figure 14-95. PCI Express CSB Bridge Control Register (PEX\_CSB\_CTRL)

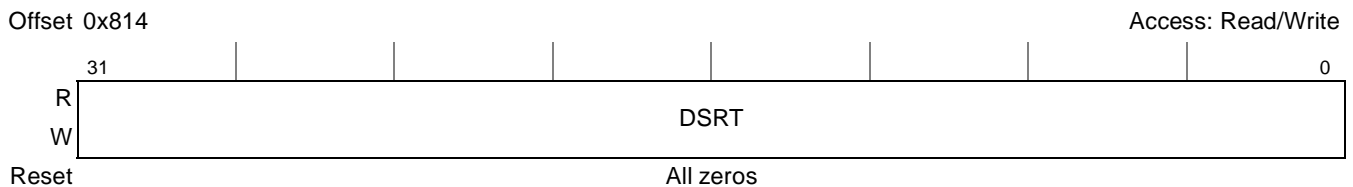
Table 14-93 defines the bit fields of PEX\_CS\_B\_CTRL.

**Table 14-93. PEX\_CS\_B\_CTRL Register Fields Description**

Bits	Name	Description
31–10	—	Reserved
9–8	DSAD	Depth of descriptors array. Indicates the number of descriptors that should be placed at a contiguous addresses block. The PCI Express controller DMA uses this information to fetch each block of descriptors by a burst transaction. The implicit address of the next descriptor is the next memory location. The last descriptor in the contiguous block contains the explicit address pointer of the next set of descriptors. See Section 14.8.4, “Descriptor-Based DMA,” for detailed description. Note that for most usages this field should be programmed to 00. 00 1—Single fetch descriptor chain mode. Each descriptor explicitly contains the address of the next descriptor. 01 2—Burst fetch descriptor chain mode. The PCI Express controller will fetch two contiguous descriptors in a burst. 10 4—Burst fetch descriptor chain mode. The PCI Express controller will fetch four contiguous descriptors in a burst. 11 Reserved
7–4	—	Reserved
3	RDMAE	Read DMA enable. Must be set to enable the read DMA operation.
2	WDMAE	Write DMA enable. Must be set to enable the write DMA operation.
1	IBPIOE	Inbound PIO enable. Must be set to enable the PCI Express Inbound PIO operation.
0	OBPIOE	Outbound PIO enable. Must be set to enable the PCI Express outbound PIO operation.

### 14.5.2.2 PCI Express DMA Descriptor Timer Register (PEX\_DMA\_DSTMR)

PEX\_DMA\_DSTMR, shown in Figure 14-96, contains the timer value the DMA engine should wait for before reading the next descriptor once it encounters a descriptor that is not ready. The timer should be programmed to allow sufficient number of clocks before the DMA tries to fetch the descriptors again.



**Figure 14-96. PCI Express DMA Descriptor Timer Register (PEX\_DMA\_DSTMR)**

Table 14-94 defines the bit field for PEX\_DMA\_DSTMR.

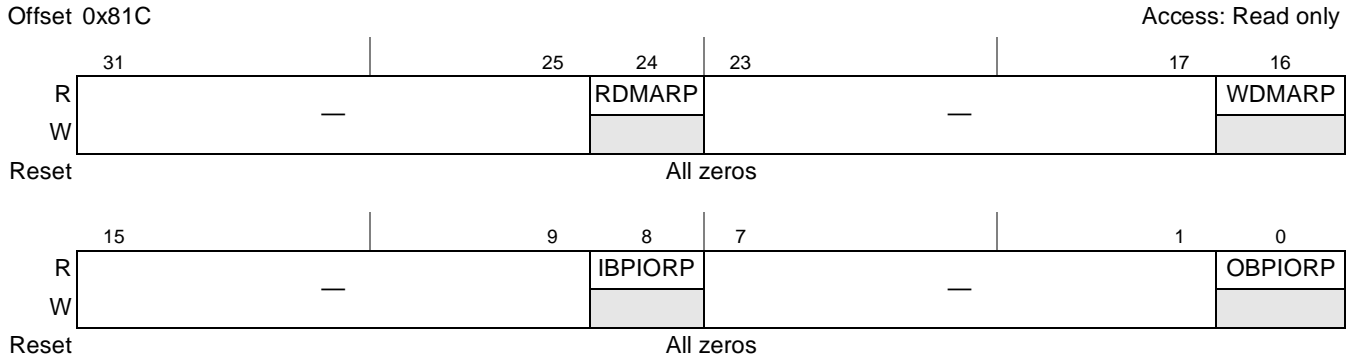
**Table 14-94. PEX\_DMA\_DSTMR Fields Description**

Bits	Name	Description
31–0	DSRT	Descriptor ready timer. Represents the number of CSB bridge clocks that the DMA engine should wait before checking whether the next descriptor is ready when it encounters invalid descriptor.



### 14.5.2.3 PCI Express CSB Bridge Status Register (PEX\_CSB\_STAT)

PEX\_CSB\_STAT, shown in Figure 14-97, maintains the activity status of the DMA and PIO transactions. When a transaction is initiated by the PCI Express DMA or PIO, the corresponding pending bit is set until a response is received.



**Figure 14-97. PCI Express CSB Bridge Status Register (PEX\_CSB\_STAT)**

Table 14-95 defines the bit field for PEX\_CSB\_STAT.

**Table 14-95. PEX\_CSB\_STAT Register Fields Description**

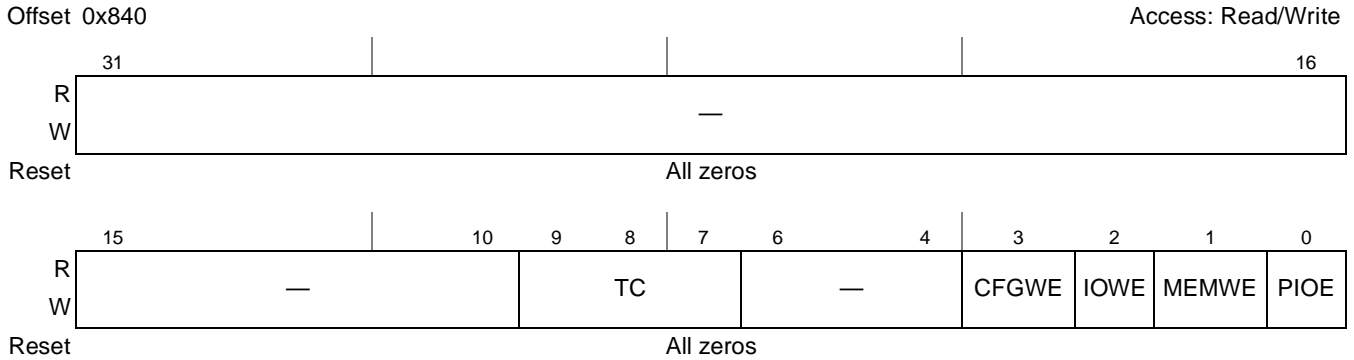
Bits	Name	Description
31–25	—	Reserved
24	RDMARP	Read DMA read transaction pending. Indicates whether a response is pending from the PCI Express bus to a transfer by the read DMA engine. 0 No response pending 1 Response is pending
23–17	—	Reserved
16	WDMARP	Write DMA read transaction pending. Indicates whether a response is pending from the CSB bus to a transfer from the write DMA engine. 0 No response pending 1 Response is pending
15–9	—	Reserved
8	IBPIORP	PCI Express inbound PIO read transaction pending. Indicates whether a response is pending from the CSB bus for an inbound transfer from the PCI Express bus 0 No response pending 1 Response is pending
7–1	—	Reserved
0	OBPIORP	PCI Express outbound PIO read transaction pending. Indicates whether a response is pending from the PCI Express bus for a transfer initiated on the CSB bus. 0 No response pending 1 Response is pending

### 14.5.3 PCI Express Outbound PIO Registers

The registers discussed in this section control PIO outbound transactions initiated by a CSB master.

### 14.5.3.1 PCI Express Outbound PIO Control Register (PEX\_CSB\_OBCTRL)

PEX\_CSB\_OBCTRL, shown in Figure 14-98, controls the PCI Express Outbound PIO operations.



**Figure 14-98. PCI Express Outbound PIO Control Register (PEX\_CSB\_OBCTRL)**

Table 14-96 defines the bit fields for PEX\_CSB\_OBCTRL.

**Table 14-96. PEX\_CSB\_OBCTRL Register Fields Description**

Bits	Name	Description
31–10	—	Reserved
9–7	TC	Traffic class. Indicates TC value to be used for TLP generation corresponding to traffic received by the CSB slave.
6–4	—	Reserved
3	CFGWE	Configuration window enable. Must be set to enable an outbound configuration transaction. Indicates that a CSB transactions directed to an outbound window can be mapped to Config write and read TLPs and transmitted to the PCI Express link.
2	IOWE	I/O window enable. Must be set to enable an outbound I/O transaction. Indicates that a CSB transactions directed to an outbound window can be mapped to I/O write and read TLPs and transmitted to the PCI Express link.
1	MEMWE	Memory window enable. Must be set to enable an outbound Memory transaction. Indicates that a CSB transactions directed to an outbound window can be mapped to Memory write and read TLPs and transmitted to the PCI Express link.
0	PIOE	PIO enable. Must be set to enable an outbound PIO transaction. This field controls the general enable of the PCI Express CSB bridge outbound PIO operation and should be set together with the other window enable fields in this register.

### 14.5.3.2 PCI Express Outbound PIO Status Register (PEX\_CSB\_OBSTAT)

PEX\_CSB\_OBSTAT, shown in Figure 14-99, maintains the status of the CSB PIO operations through the CSB slave controller. This register is replicated for each CSB PIO engine implemented.



Figure 14-99. PCI Express Outbound PIO Status Register (PEX\_CSB\_OBSTAT)

Table 14-97 defines the bit fields for PEX\_CSB\_OBSTAT.

Table 14-97. PEX\_CSB\_OBSTAT Register Fields Description

Bit	Name	Description
31–4	—	Reserved
3	PEXER	PCI Express error. Hardware sets this bit to indicate that an outbound PIO operation could not be completed successfully because there was an error in PCI Express completion received.
2	CSBER	CSB Bridge error. Hardware sets this bit to indicate that an error has occurred during a PIO outbound access to the CSB bridge by a CSB master.
1	BMPER	Bridge mapping error. Hardware sets this bit to indicate that an outbound PIO operation could not be completed successfully because a CSB bridge address mapping error has occurred.
0	BENER	Bridge enable error. Hardware sets this bit to indicate that the an outbound PIO operation could not be completed successfully because the CSB bridge outbound PIO operation was not enabled.

### 14.5.4 PCI Express Inbound PIO Registers

The registers discussed in this section control PIO inbound transactions initiated by a PCI Express device.

### 14.5.4.1 PCI Express Inbound PIO Control Register (PEX\_CSIBCTRL)

PEX\_CSIBCTRL, shown in Figure 14-100, controls the PCI Express inbound PIO operations.

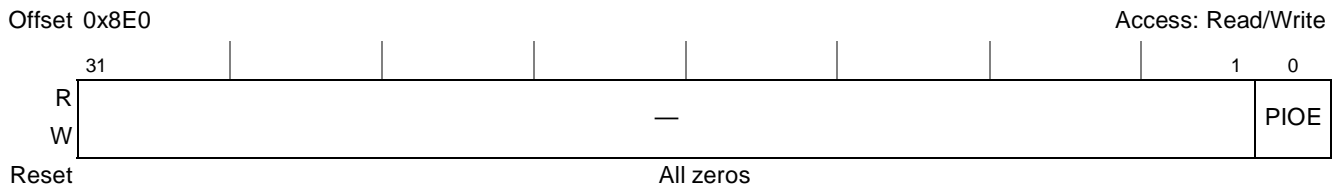


Figure 14-100. PCI Express Inbound PIO Control Register (PEX\_CSIBCTRL)

Table 14-98 defines the bit fields for PEX\_CSIBCTRL.

Table 14-98. PEX\_CSIBCTRL Register Fields Description

Bit	Name	Description
31–1	—	Reserved
0	PIOE	PIO enable. Must be set to enable an inbound PIO transaction. This field controls the general enable of the PCI Express CSB bridge inbound PIO operation.

### 14.5.4.2 PCI Express Inbound PIO Status Register (PEX\_CSIBSTAT)

PEX\_CSIBSTAT, shown in Figure 14-101, maintains the status of PCI Express Inbound PIO operations through the PCI Express CSB bridge.

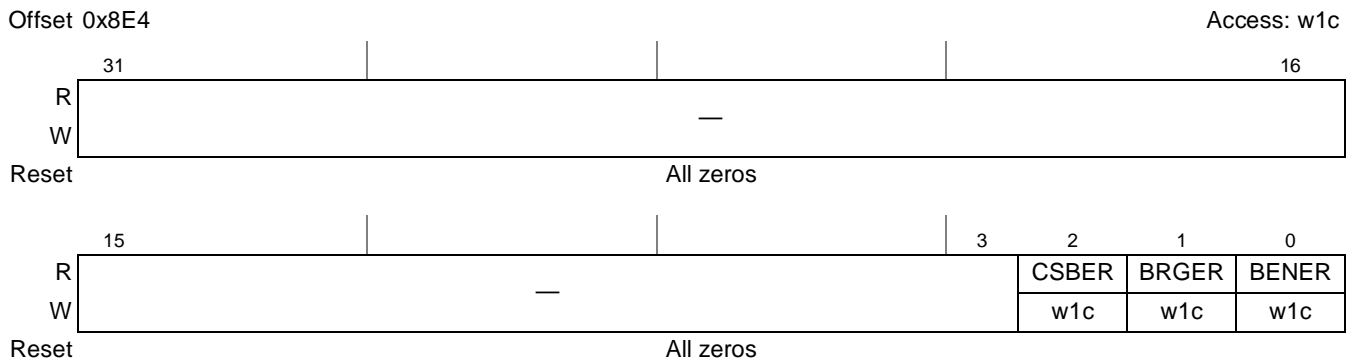


Figure 14-101. PCI Express Inbound PIO Status Register (PEX\_CSIBSTAT)

Table 14-99 defines the bit fields for PEX\_CSIBSTAT.

Table 14-99. PEX\_CSIBSTAT Register Fields Description

Bit	Name	Description
31–3	—	Reserved
2	CSBER	CSB Bus Error. Hardware sets this bit to indicate that a CSB transaction bus error was encountered during an inbound PIO operation.

Table 14-99. PEX\_CSB\_IBSTAT Register Fields Description (continued)

Bit	Name	Description
1	BRGER	CSB Bridge Error. Hardware sets this bit to indicate that an inbound PIO operation cannot complete successfully because of a CSB bridge error.
0	BENER	Bridge enable error. Hardware sets this bit to indicate that the an inbound PIO operation cannot complete successfully because the CSB bridge inbound PIO operation is not enabled.

## 14.5.5 DMA Registers

This section describes the following registers:

- Section 14.5.5.1, “PCI Express Write DMA Control Register (PEX\_WDMA\_CTRL)”
- Section 14.5.5.2, “PCI Express Write DMA First Address Register (PEX\_WDMA\_ADDR)”
- Section 14.5.5.3, “PCI Express Write DMA Status Register (PEX\_WDMA\_STAT)”
- Section 14.5.5.4, “PCI Express Read DMA Control Register (PEX\_RDMA\_CTRL)”
- Section 14.5.5.5, “PCI Express Read DMA First Address Register (PEX\_RDMA\_ADDR)”
- Section 14.5.5.6, “PCI Express Read DMA Status Register (PEX\_RDMA\_STAT)”

### 14.5.5.1 PCI Express Write DMA Control Register (PEX\_WDMA\_CTRL)

PEX\_WDMA\_CTRL, shown in Figure 14-102, controls the WDMA operations.

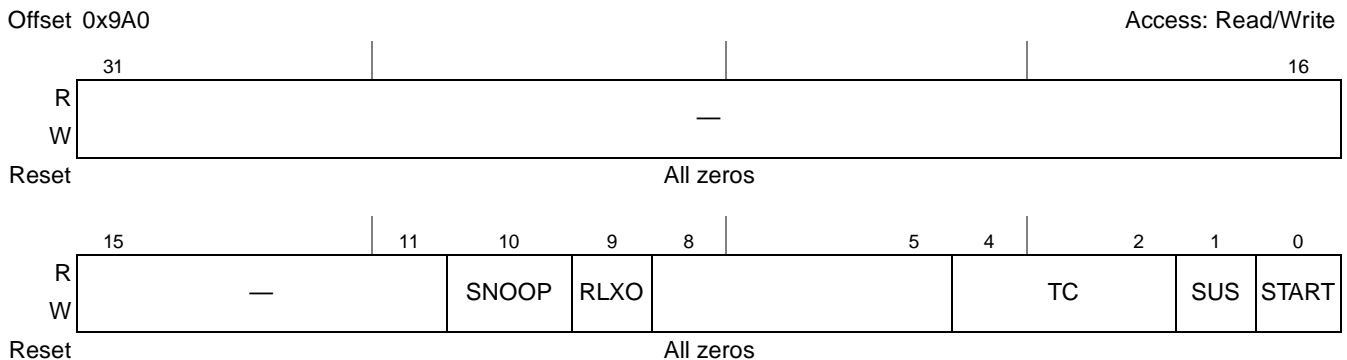


Figure 14-102. PCI Express Write DMA Control Register (PEX\_WDMA\_CTRL)

Table 14-100 defines the bit fields of PEX\_WDMA\_CTRL.

Table 14-100. PEX\_WDMA\_CTRL Register Fields Description

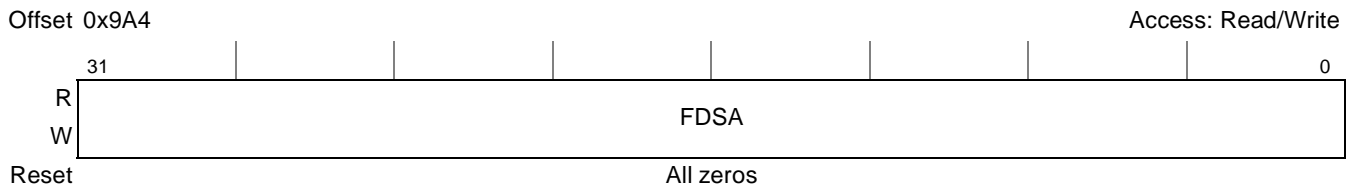
Bit	Name	Description
31–11	—	Reserved
10	SNOOP	Snoop for write and read transactions to the descriptor. Controls the snooping of the e300 core on the CSB bus to a transaction initiated by the WDMA. 0 Snoop disabled. 1 Snoop enabled.
9	RLXO	Relaxed ordering for PCI Express. Indicates the relaxed ordering bit to be used for all PCI Express transactions initiated by the DMA controller.

**Table 14-100. PEX\_WDMA\_CTRL Register Fields Description (continued)**

Bit	Name	Description
8–5	—	Reserved.
4–2	TC	Traffic Class. Indicates the traffic class value to be used for TLP generation corresponding to the traffic generated by the DMA.
1	SUS	DMA suspend. Software sets this bit to suspend the DMA controller.
0	START	DMA start. Software should set this bit to indicate that a descriptor is ready and the DMA controller can start transmission. Hardware resets this bit when the descriptor fetch cycle is initiated.

### 14.5.5.2 PCI Express Write DMA First Address Register (PEX\_WDMA\_ADDR)

PEX\_WDMA\_ADDR, shown in [Figure 14-103](#), contains the CSB local memory address of the first descriptor. Note that the content is byte swapped from a CSB native address.



**Figure 14-103. PCI Express Write DMA First Address Register (PEX\_WDMA\_ADDR)**

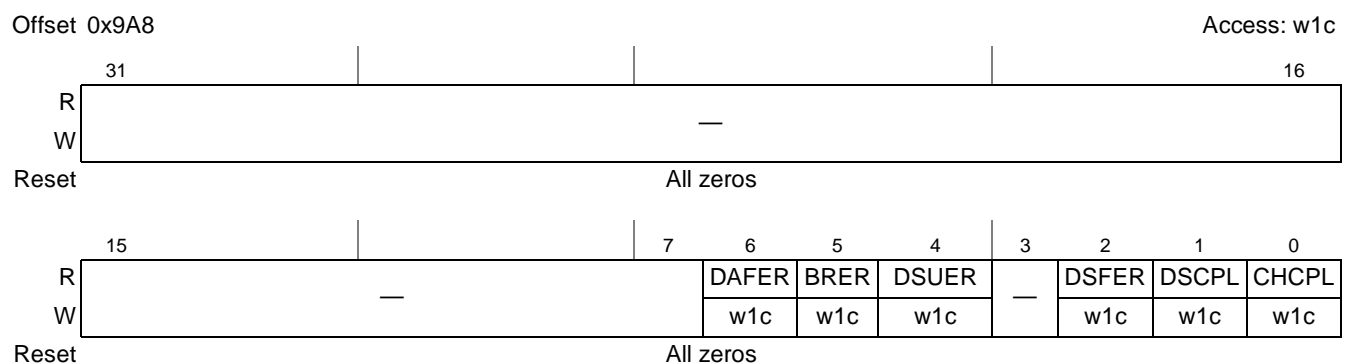
[Table 14-101](#) defines the bit fields of PEX\_WDMA\_ADDR.

**Table 14-101. PEX\_WDMA\_ADDR Register Fields Description**

Bit	Name	Description
31–0	FDSA	First descriptor address. Indicates the address of the first descriptor on the CSB local memory (byte-swapped).

### 14.5.5.3 PCI Express Write DMA Status Register (PEX\_WDMA\_STAT)

PEX\_WDMA\_STAT, shown in [Figure 14-104](#), maintains the status of write DMA operations.



**Figure 14-104. PCI Express Write DMA Status Register (PEX\_WDMA\_STAT)**

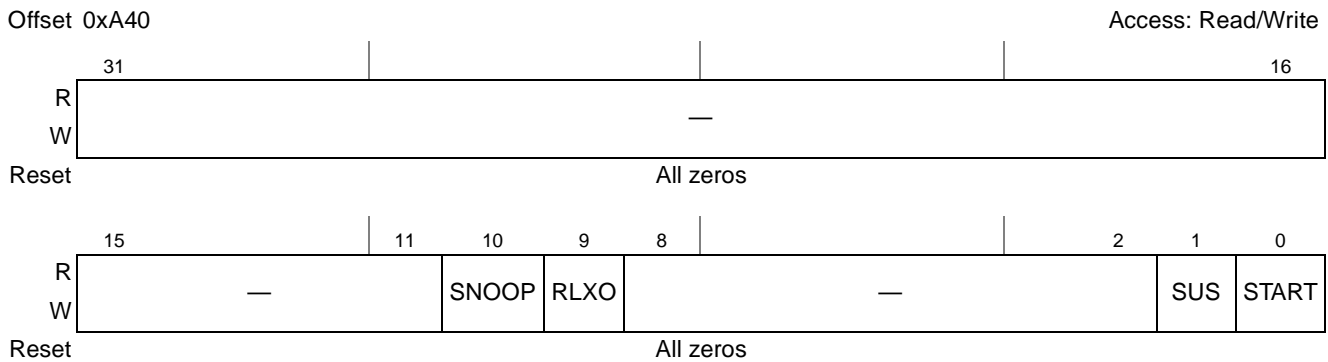
[Table 14-102](#) defines the bit fields of PEX\_WDMA\_STAT.

**Table 14-102. PEX\_WDMA\_STAT Register Fields Description**

Bit	Name	Description
31–7	—	Reserved
6	DAFER	DMA data fetch error. Hardware sets this bit to indicate an error during the data fetch operation.
5	BRER	Bridge error. Hardware sets this bit to indicate that DMA operation cannot complete successfully because of a CSB bridge error.
4	DSUER	Descriptor update error. Hardware sets this bit to indicate an error during descriptor update operation.
3	—	Reserved
2	DSFER	Descriptor fetch error. This bit is set by hardware to indicate that a descriptor read from the CSB has terminated with an error.
1	DSCPL	Descriptor DMA transfer completed. Hardware sets this bit after completing the transaction for the descriptor.
0	CHCPL	DMA chain transfer completed. Hardware sets this bit after completing the transaction in all the descriptors that are currently programmed. This bit is set when DMA operation is complete and the DMA controller encounters a NULL descriptor. <b>Note:</b> When hardware sets this bit it is not guaranteed that the transferred data has fully reached its final destination. Software should guarantee this another way. For additional information see the PEX2 erratum in the errata document of the device.

#### 14.5.5.4 PCI Express Read DMA Control Register (PEX\_RDMA\_CTRL)

PEX\_RDMA\_CTRL, shown in Figure 14-105, controls the RDMA operations.



**Figure 14-105. PCI Express Read DMA Control Register (PEX\_RDMA\_CTRL)**

Table 14-103 defines the bit fields of PEX\_RDMA\_CTRL.

**Table 14-103. PEX\_RDMA\_CTRL Register Fields Description**

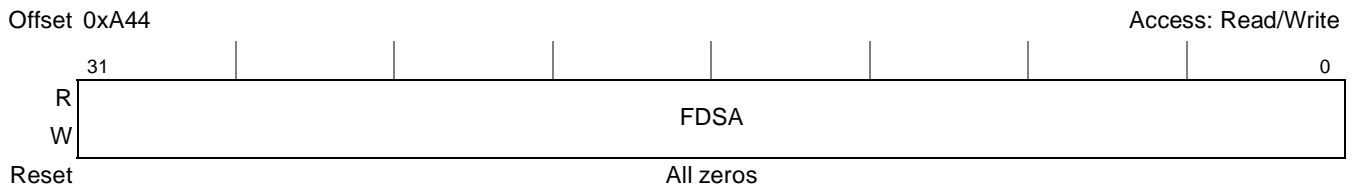
Bits	Name	Description
31–11	—	Reserved
10	SNOOP	Snoop for write and read transactions to the descriptor. Controls the snooping of the e300 core on the CSB bus to a transaction initiated by the RDMA. 0 Snoop disabled. 1 Snoop enabled.

**Table 14-103. PEX\_RDMA\_CTRL Register Fields Description (continued)**

Bits	Name	Description
9	RLXO	Relaxed ordering for PCI Express. Indicates the relaxed ordering bit to be used for all PCI Express transactions initiated by the DMA controller.
8–2	—	Reserved
1	SUS	DMA suspend. Software sets this bit to suspend the DMA controller.
0	START	DMA start. Software can set this bit to indicate that a descriptor is ready and the DMA controller can start transmission. Hardware resets this bit when a descriptor fetch cycle is initiated.

### 14.5.5.5 PCI Express Read DMA First Address Register (PEX\_RDMA\_ADDR)

PEX\_RDMA\_ADDR, shown in [Figure 14-106](#), contains the local memory address of the first descriptor. Note that the content is byte swapped from a CSB native address.



**Figure 14-106. PCI Express Read DMA First Address Register (PEX\_RDMA\_ADDR)**

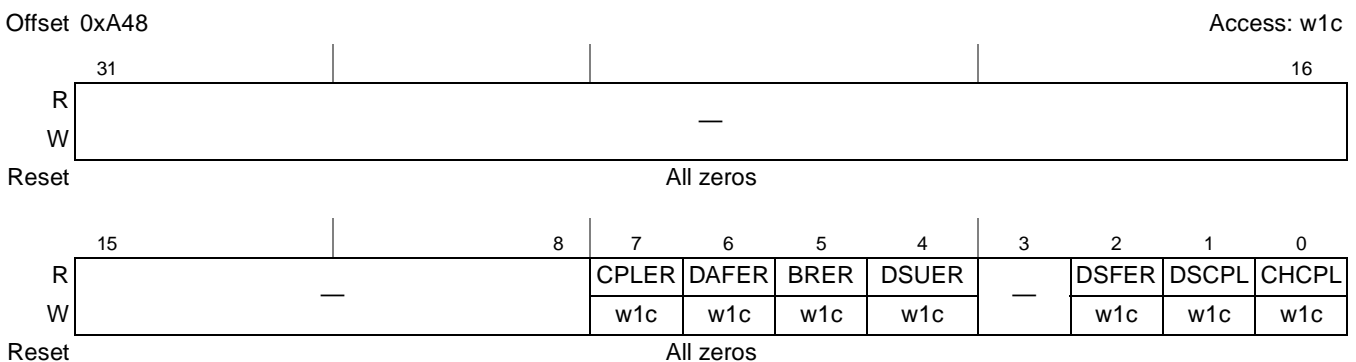
[Table 14-104](#) defines the bit fields of PEX\_RDMA\_ADDR.

**Table 14-104. PEX\_RDMA\_ADDR Register Fields Description**

Bits	Name	Description
31–0	FDSA	First descriptor address. Indicates the address of the first descriptor on the CSB local memory (byte swapped).

### 14.5.5.6 PCI Express Read DMA Status Register (PEX\_RDMA\_STAT)

PEX\_RDMA\_STAT, shown in [Figure 14-107](#), maintains the status of read DMA operations.



**Figure 14-107. PCI Express Read DMA Status Register (PEX\_RDMA\_STAT)**

[Table 14-105](#) defines the bit fields of PEX\_RDMA\_STAT.



**Table 14-105. PEX\_RDMA\_STAT Register Fields Description**

Bits	Name	Description
31–8	—	Reserved
7	CPLER	PCI Express completion error. Hardware sets this bit to indicate that DMA operation cannot complete successfully because of a PCI Express error.
6	DAFER	DMA data write error. The hardware sets this bit to indicate an error during data write operation.
5	BRER	Bridge error. Hardware sets this bit to indicate that DMA operation cannot complete successfully because of a CSB bridge error.
4	DSUER	Descriptor update error. Hardware sets this bit to indicate an error during descriptor update operation.
3	—	Reserved
2	DSFER	Descriptor fetch error. This bit is set by hardware to indicate that a descriptor read from the CSB has terminated with an error.
1	DSCPL	Descriptor DMA transfer completed. Hardware sets this bit after completing the transaction for the descriptor.
0	CHCPL	DMA chain transfer completed. Hardware sets this bit after completing the transaction in all the descriptors that are currently programmed. This bit is set when DMA operation is complete and the DMA controller encounters a NULL descriptor. <b>Note:</b> When hardware sets this bit it is not guaranteed that the transferred data has fully reached its final destination. Software should guarantee this another way.

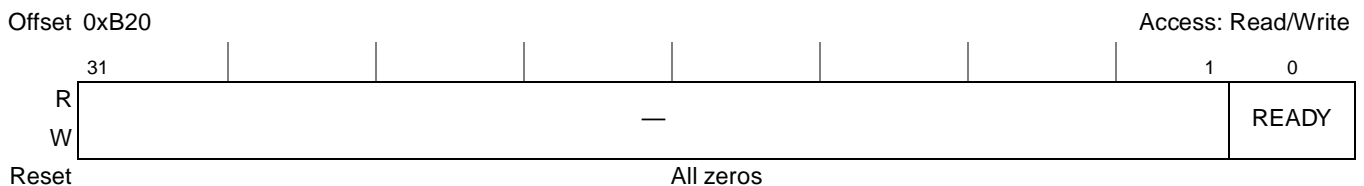
### 14.5.6 Mailbox Registers

This section describes the following registers:

- [Section 14.5.6.1, “PCI Express Outbound Mailbox Control Register \(PEX\\_OMBCR\)”](#)
- [Section 14.5.6.2, “PCI Express Outbound Mailbox Data Register \(PEX\\_OMBDR\)”](#)
- [Section 14.5.6.3, “PCI Express Inbound Mailbox Control Register \(PEX\\_IMBCR\)”](#)
- [Section 14.5.6.4, “PCI Express Inbound Mailbox Data Register \(PEX\\_IMBDR\)”](#)

#### 14.5.6.1 PCI Express Outbound Mailbox Control Register (PEX\_OMBCR)

PEX\_OMBCR, shown in [Figure 14-108](#), controls the generation of an outbound interrupt from the CSB local host to the PCI Express and indicates that the local host has programmed the data mailbox register, and it is ready to be read. Setting the ready bit generates an interrupt to the PCI Express host if enabled. The PCI Express host should clear the ready bit after reading the data mailbox register.



**Figure 14-108. PCI Express Outbound Mailbox Control Register (PEX\_OMBCR)**

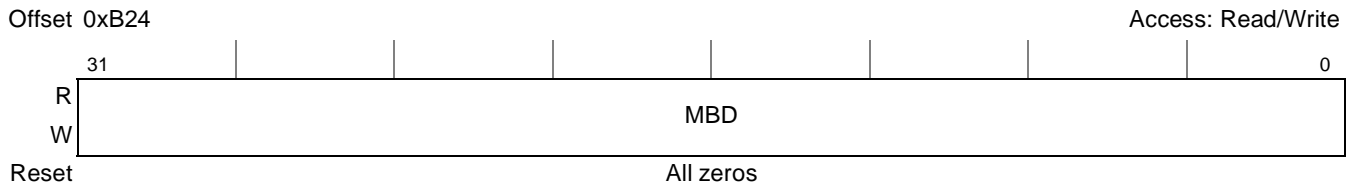
[Table 14-106](#) defines the bit fields of PEX\_OMBCR.

**Table 14-106. PEX\_OMBCR Register Fields Description**

Bits	Name	Description
31–1	—	Reserved
0	READY	Outbound mailbox ready. If set, indicates that mailbox has valid data to be read by the PCI Express host and generates an interrupt if enabled.

### 14.5.6.2 PCI Express Outbound Mailbox Data Register (PEX\_OMBDR)

PEX\_OMBDR, shown in [Figure 14-109](#), contains the data to be read by the PCI Express host when it receives an interrupt.



**Figure 14-109. MPC Express Outbound Mailbox Data Register (PEX\_OMBDR)**

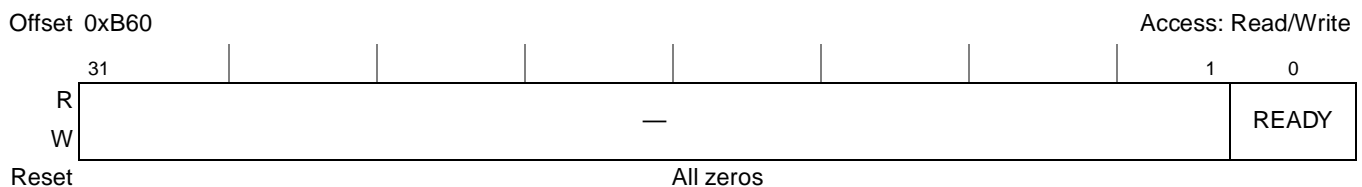
[Table 14-107](#) defines the bit fields of PEX\_OMBDR.

**Table 14-107. PEX\_OMBDR Register Fields Description**

Bits	Name	Description
31–0	MBD	Mailbox Data. Contains the data to be read by the PCI Express host upon receiving an interrupt.

### 14.5.6.3 PCI Express Inbound Mailbox Control Register (PEX\_IMBCR)

PEX\_IMBCR, shown in [Figure 14-110](#), controls the generation of an interrupt to the local host indicating that the PCI Express host has programmed the data mailbox register, and it is ready to be read. The CSB host clears the bit after reading out the mailbox register. Note that setting the ready bit generates an interrupt to the CSB host if enabled. The CSB host should clear the ready bit after reading the data mailbox register.



**Figure 14-110. PCI Express Inbound Mailbox Control Register (PEX\_IMBCR)**

[Table 14-108](#) defines the bit fields of PEX\_IMBCR.

**Table 14-108. PEX\_IMBCR Register Fields Description**

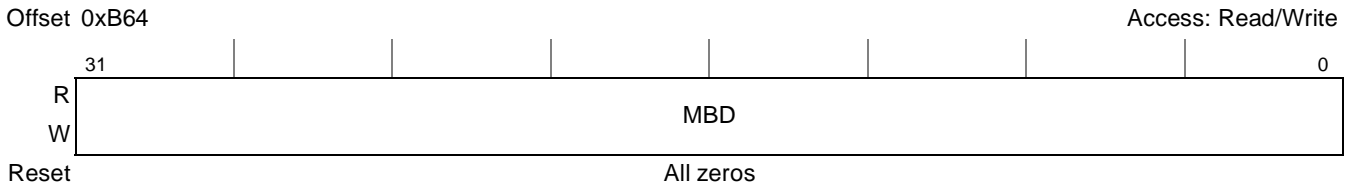
Bits	Name	Description
------	------	-------------

**Table 14-108. PEX\_IMBCR Register Fields Description (continued)**

31–1	—	Reserved
0	READY	Inbound mailbox ready. If set, indicates that mailbox has valid data to be read by the CSB local host and generates an interrupt if enabled.

### 14.5.6.4 PCI Express Inbound Mailbox Data Register (PEX\_IMBDR)

PEX\_IMBDR, shown in Figure 14-111, contains the data to be read by the local CSB host.



**Figure 14-111. PCI Express Inbound Mailbox Data Register (PEX\_IMBDR)**

Table 14-109 defines the bit fields of PEX\_IMBDR.

**Table 14-109. PEX\_IMBDR Register Fields Description**

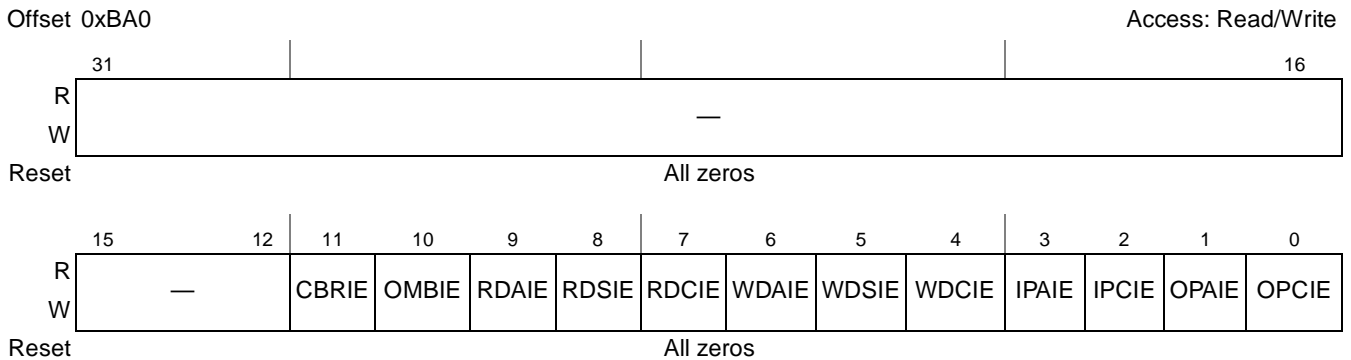
Bits	Name	Description
31–0	MBD	Mailbox data. Contains the data to be read by the CSB local host upon receiving an interrupt.

## 14.5.7 PCI Express Host Interrupt Registers

This section describes the registers for generating interrupts to the PCI Express host. It consists of interrupt status registers and enable registers. Interrupts are generated only if the corresponding enable bit is set. The device supports generation of MSI interrupts. Using these registers to generate interrupts to the PCI Express host is applicable for only PCI Express EP applications.

### 14.5.7.1 PCI Express Host Interrupt Enable Register (PEX\_HIER)

PEX\_HIER, shown in Figure 14-112, enables the generation of interrupts to the PCI Express host at various events during the CBS bridge, PIO and DMA operation.



**Figure 14-112. PCI Express Host Interrupt Enable Register (PEX\_HIER)**

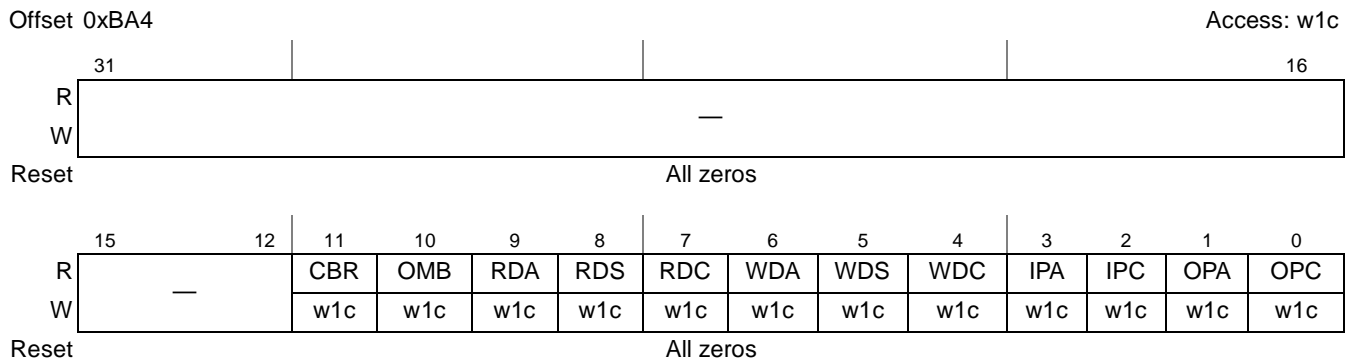
Table 14-110 defines the bit fields of PEX\_HIER.

**Table 14-110. PEX\_HIER Register Fields Description**

Bits	Name	Description
31–12	—	Reserved
11	CBRIE	CSB bridge reset interrupt enable. If set, enables the generation of an interrupt upon a CSB bridge reset.
10	OMBIE	Outbound mailbox ready interrupt enable. If set, enables the generation of interrupt when the outbound mailbox control register ready bit (PEX_OMBCR[READY]) is set.
9	RDAIE	RDMA transfer aborted interrupt enable. If set, enables the generation of interrupt for every Read DMA transaction aborted.
8	RDSIE	RDMA descriptor transfer completed interrupt enable. If set, enables the generation of an interrupt when a read DMA transaction corresponding to a descriptor successfully completes.
7	RDCIE	RDMA chain descriptor transfer completed interrupt enable. If set, enables the generation of interrupt when a read DMA transaction for an end-of-descriptor successfully completes.
6	WDAIE	WDMA transfer aborted interrupt enable. If set, enables the generation of interrupt for every DMA transaction aborted.
5	WDSIE	WDMA descriptor transfer completed interrupt enable. If set, enables the generation of interrupt when DMA transactions corresponding to a descriptor successfully complete.
4	WDCIE	WDMA chain descriptor transfer completed interrupt enable. If set, enables the generation of interrupt when DMA transactions for end-of-descriptor successfully complete.
3	IPAIE	Inbound PIO transaction aborted interrupt enable. If set, enables the generation of an interrupt for every inbound PCI Express PIO transaction aborted.
2	IPCIE	Inbound PIO transaction completed interrupt enable. If set, enables the generation of an interrupt for every inbound PCI Express PIO transaction that successfully completes.
1	OPAIE	Outbound PIO transaction abort interrupt enable. If set, enables the generation of an interrupt for every outbound PIO transaction aborted.
0	OPCIE	Outbound PIO transaction completed interrupt enable. If set, enables the generation of an interrupt for every outbound PIO transaction that successfully completes.

### 14.5.7.2 PCI Express Host Interrupt Status Register (PEX\_HISR)

PEX\_HISR, shown in Figure 14-113, maintains the status for interrupts issued to the PCI Express host.



**Figure 14-113. PCI Express Host Interrupt Status Register (PEX\_HISR)**



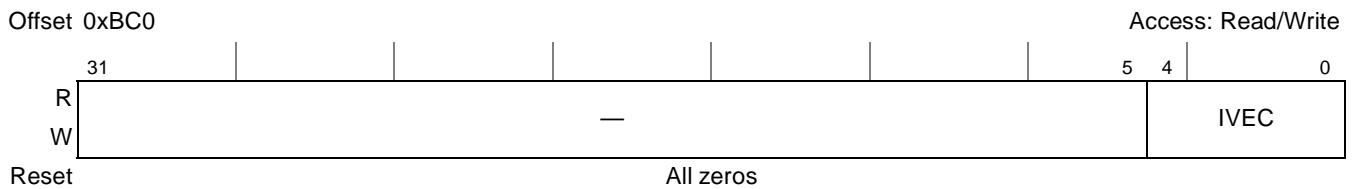
Table 14-112 defines the bit fields of PEX\_HOPIVR.

**Table 14-112. PEX\_HOPIVR Register Fields Description**

Bits	Name	Description
31–5	—	Reserved
4–0	IVEC	Interrupt Vector. Contains the vector value for MSI.

#### 14.5.7.4 PCI Express Host Inbound PIO Interrupt Vector Register (PEX\_HIPIVR)

PEX\_HIPIVR, shown in Figure 14-115, contains the interrupt vector for MSI interrupt generation upon an inbound PIO event. This vectors will be sent by the MSI interrupts to the PCI Express host if the interrupt is enabled.



**Figure 14-115. PCI Express Host Inbound PIO Interrupt Vector Register (PEX\_HIPIVR)**

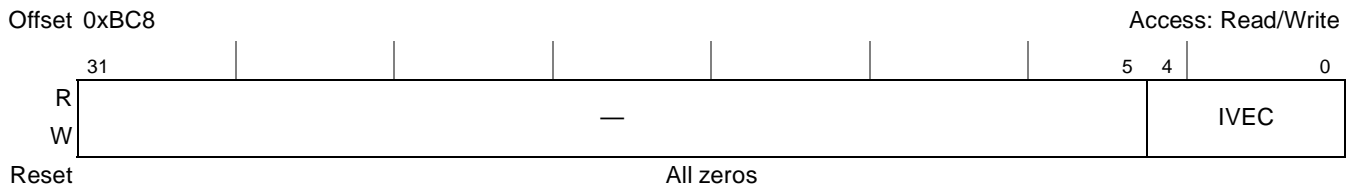
Table 14-113 defines the bit fields of PEX\_HIPIVR.

**Table 14-113. PEX\_HIPIVR Register Fields Description**

Bits	Name	Description
31–5	—	Reserved
4–0	IVEC	Interrupt vector. Contains the vector value for MSI.

#### 14.5.7.5 PCI Express Host Write DMA Interrupt Vector Register (PEX\_HWDIVR)

PEX\_HWDIVR, shown in Figure 14-116, contains the interrupt vector for an MSI interrupt issued to the PCI Express host upon WDMA events.



**Figure 14-116. PCI Express Host Write DMA Interrupt Vector Register (PEX\_HWDIVR)**



## 14.5.8 CSB System Interrupt Registers

This section describes the registers for generating interrupts to the CSB system host (through the IPIC). It consists of interrupt status registers and enable registers. Interrupts are generated only if the corresponding enable bit is set, upon PIO, DMA and Miscellaneous events. The user has the flexibility of combining all or partial interrupt signals to generate interrupts to the CSB system host.

### 14.5.8.1 CSB System PIO Interrupt Enable Register (PEX\_CSPIER)

PEX\_CSPIER, shown in Figure 14-119, controls the generation of interrupt to the CSB processor at various events during an CSB or PCI Express PIO operation.

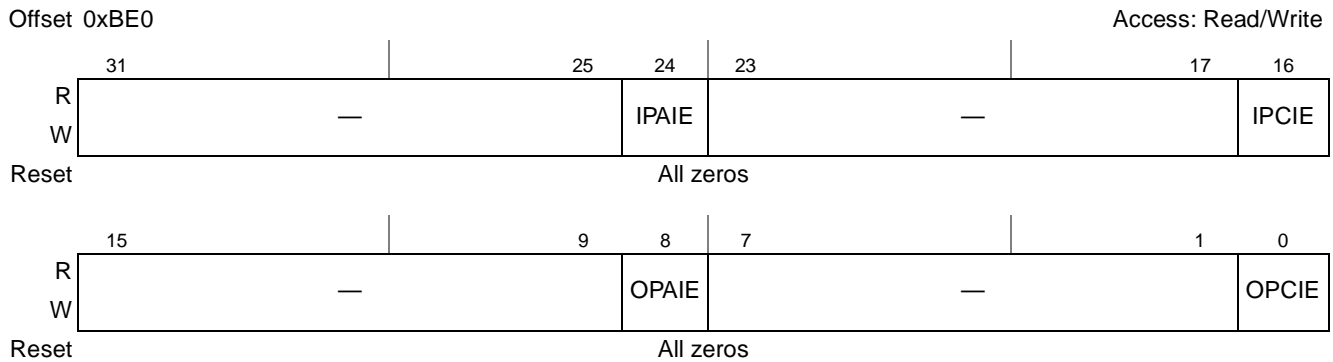


Figure 14-119. CSB System PIO Interrupt Enable Register (PEX\_CSPIER)

Table 14-117 defines the bit fields of PEX\_CSPIER.

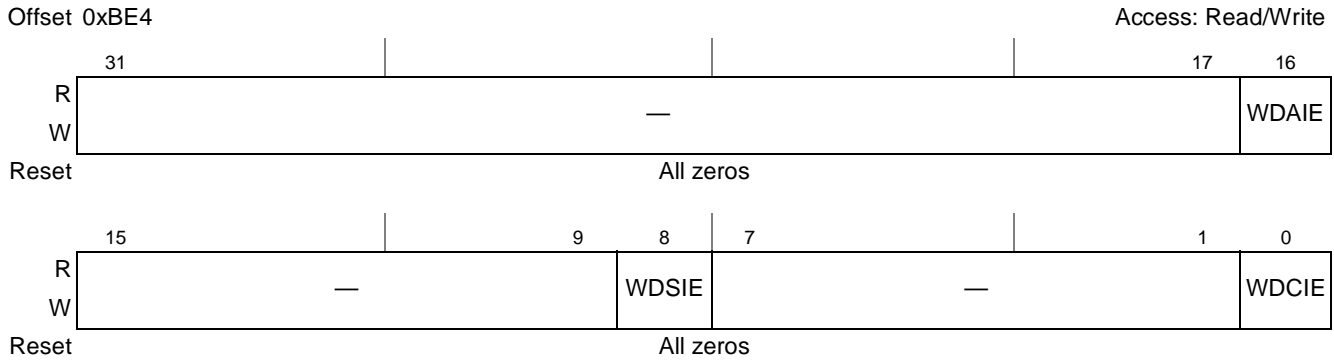
Table 14-117. PEX\_CSPIER Register Fields Description

Bit	Name	Description
31–25	—	Reserved
24	IPAIE	Inbound PIO transaction aborted interrupt enable. If set, enables the generation of an interrupt for every inbound PCI Express PIO transaction aborted.
23–17	—	Reserved
16	IPCIE	Inbound PIO transaction completed interrupt enable. If set, enables the generation of an interrupt for every inbound PCI Express PIO transaction that successfully completes.
15–9	—	Reserved
8	OPAIE	Outbound PIO transaction abort interrupt enable. If set, enables the generation of an interrupt for every outbound PIO transaction aborted.
7–1	—	Reserved
0	OPCIE	Outbound PIO transaction completed interrupt enable. If set, enables the generation of an interrupt for every outbound PIO transaction that successfully completes.

### 14.5.8.2 CSB System Write DMA Interrupt Enable Register (PEX\_CSWDIER)

PEX\_CSWDIER, shown in Figure 14-120, controls the generation of interrupt to the CSB processor at various events during a WDMA operation.





**Figure 14-120. CSB System Write DMA Interrupt Enable Register (PEX\_CSWDIER)**

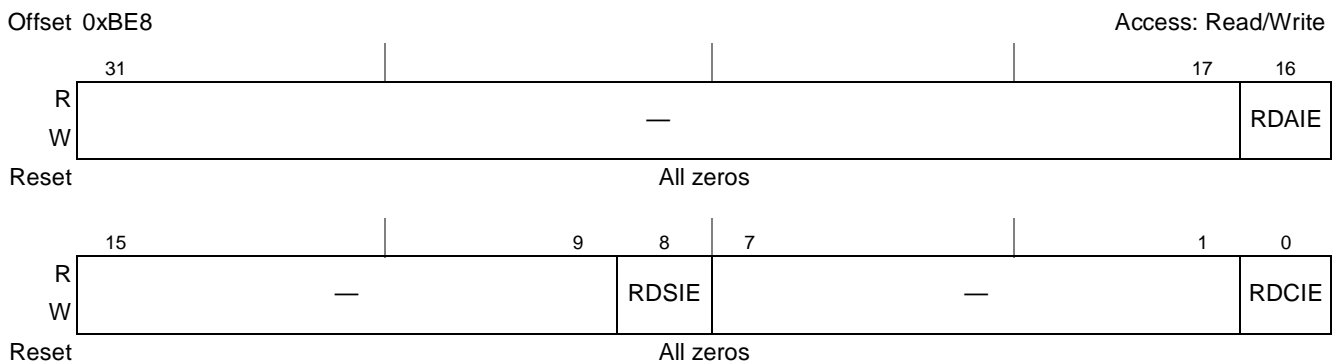
Table 14-118 defines the bit fields of PEX\_CSWDIER.

**Table 14-118. PEX\_CSWDIER Register Fields Description**

Bit	Name	Description
31–17	—	Reserved
16	WDAIE	WDMA transfer aborted interrupt enable. If set, enables the generation of interrupt for every DMA transaction aborted.
15–9	—	Reserved
8	WDSIE	WDMA descriptor transfer completed interrupt enable. If set, enables the generation of interrupt when DMA transactions corresponding to a descriptor complete successfully.
7–1	—	Reserved
0	WDCIE	WDMA chain descriptor transfer completed interrupt enable. If set, enables the generation of interrupt when DMA transactions for end-of-descriptor complete successfully.

### 14.5.8.3 CSB System Read DMA Interrupt Enable Register (PEX\_CSRDIER)

PEX\_CSRDIER, shown in Figure 14-121, controls the generation of interrupt to the CSB system host at various events during a RDMA operation.



**Figure 14-121. CSB System Read DMA Interrupt Enable Register (PEX\_CSRDIER)**

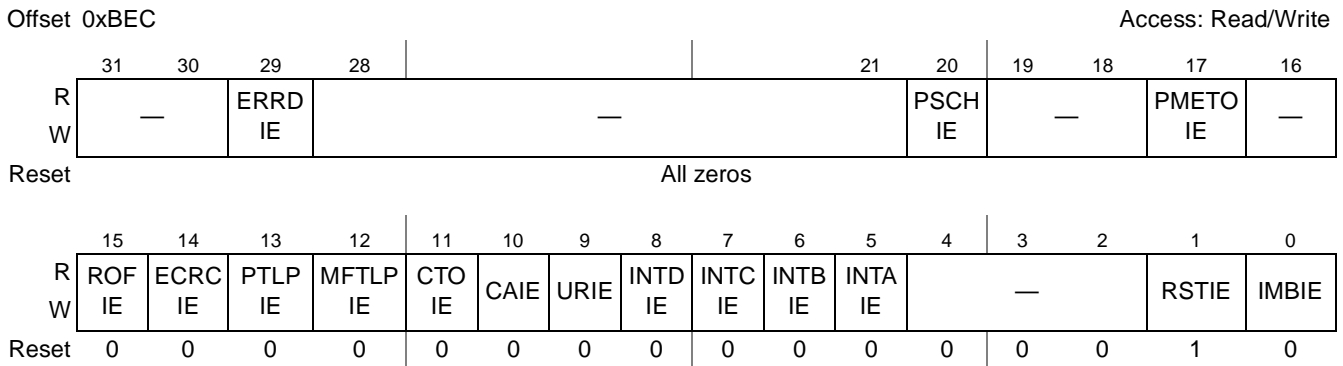
Table 14-119 defines the bit fields of PEX\_CSVDIER.

**Table 14-119. PEX\_CSVDIER Register Fields Description**

Bit	Name	Description
31–17	—	Reserved
16	RDAIE	RDMA transfer aborted interrupt enable. If set, enables the generation of interrupt for every Read DMA transaction aborted.
15–9	—	Reserved.
8	RDSIE	RDMA descriptor transfer completed interrupt enable. If set, enables the generation of an interrupt when a Read DMA transactions corresponding to a descriptor complete successfully.
7–1	—	Reserved.
0	RDCIE	RDMA chain descriptor transfer completed interrupt enable. If set, enables the generation of interrupt when a Read DMA transactions for end-of-descriptor complete successfully.

#### 14.5.8.4 CSB System Miscellaneous Interrupt Enable Register (PEX\_CSMIER)

PEX\_CSMIER, shown in Figure 14-122, controls the generation of interrupt to the CSB processor at various events.



**Figure 14-122. CSB System Miscellaneous Interrupt Enable Register (PEX\_CSMIER)**

Table 14-120 defines the bit fields of PEX\_CSMIER.

**Table 14-120. PEX\_CSMIER Register Fields Description**

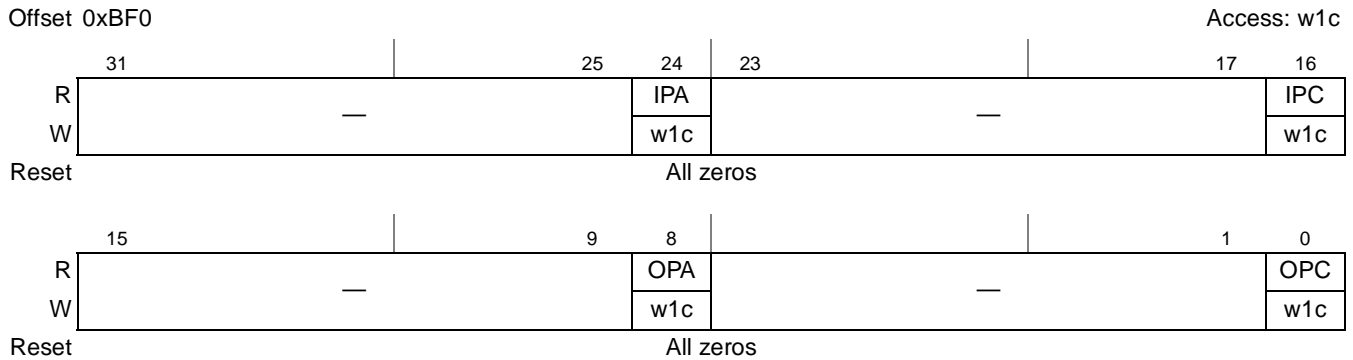
Bit	Name	Description
31–30	—	Reserved
29	ERRDIE	Error detected interrupt enable. If set, enables the generation of an interrupt when a PCI Express event occurs. The PCI Express event is reported by the Secondary status register (PCI Express Secondary Status Register) at address 0x901E. Note that a secondary mask exist by the PCI Express PCI Interrupt Mask Register (PEX_SS_INTR_MASK) at address 0x95A0. Valid only for RC.
28–21	—	Reserved
20	PSCHIE	Power state change interrupt enable. If set, enables the generation of an interrupt when a device power state change occurs.
19–18	—	Reserved

Table 14-120. PEX\_CSMIER Register Fields Description (continued)

Bit	Name	Description
17	PMETOIE	PME to Ack timeout event interrupt enable. If set, enables the generation of an interrupt when a PME to Ack timeout occurs. This bit is valid only in RC mode.
16	—	Reserved
15	ROFIE	Receive overflow error interrupt enable. If set, enables the generation of an interrupt when PCI Express reports receive overflow error.
14	ECRCIE	ECRC error interrupt enable. If set, enables the generation of an interrupt when a TLP is received by PCI Express and it fails ECRC check.
13	PTLPIE	Poisoned TLP interrupt enable. If set, enables the generation of an interrupt when a poisoned TLP is received by PCI Express.
12	MFTLPIE	Malformed TLP interrupt enable. If set, enables the generation of an interrupt when a malformed TLP is received by PCI Express
11	CTOIE	Completion timeout interrupt enable. If set, enables the generation of an interrupt when PCI Express completion timeout occurs.
10	CAIE	Completer abort interrupt enable. If set, enables the generation of an interrupt when PCI Express completion Abort is received.
9	URIE	Unsupported request interrupt enable. If set, enables the generation of an interrupt when an unsupported request is received.
8	INTDIE	PCI Express INTD interrupt enable. If set, enables the generation of an interrupt when an INTD interrupt is received on the PCI Express link. Valid for RC applications only.
7	INTCIE	PCI Express INTC interrupt enable. If set, enables the generation of an interrupt when an INTC interrupt is received on the PCI Express link. Valid for RC applications only.
6	INTBIE	PCI Express INTB interrupt enable. If set, enables the generation of an interrupt when an INTB interrupt is received on the PCI Express link. Valid for RC applications only.
5	INTAIE	PCI Express INTA interrupt enable. If set, enables the generation of an interrupt when an INTA interrupt is received on the PCI Express link. Valid for RC applications only.
4–2	—	Reserved
1	RSTIE	PCI Express reset interrupt enable. If set, enables the generation of an interrupt when PCI Express is reset.
0	IMBIE	Inbound mailbox ready interrupt enable. If set, enables the generation of interrupt whenever the inbound mailbox control register ready bit (PEX_IMBCR[READY]) is set.

### 14.5.8.5 CSB System PIO Interrupt Status Register (PEX\_CSPISR)

PEX\_CSPISR, shown in Figure 14-123, maintains the status for interrupts issued to the CSB system host related to PIO operation.



**Figure 14-123. CSB System PIO Interrupt Status Register (PEX\_CSPISR)**

Table 14-121 defines the bit fields of PEX\_CSPISR

**Table 14-121. PEX\_CSPISR Register Fields Description**

Bit	Name	Description
31–25	—	Reserved
24	IPA	Inbound PIO transaction aborted. Indicates that an inbound PCI Express PIO transaction was aborted.
23–17	—	Reserved
16	IPC	Inbound PIO transaction completed. Indicates that an inbound PCI Express PIO transaction was successfully completes.
15–9	—	Reserved
8	OPA	Outbound PIO transaction aborted. Indicates that an outbound PIO transaction was aborted.
7–1	—	Reserved
0	OPC	Outbound PIO transaction completed. Indicates that an outbound PIO transaction was successfully completes.

### 14.5.8.6 CSB System Write DMA Interrupt Status Register (PEX\_CSWDISR)

PEX\_CSWDISR, shown in Figure 14-124, maintains the status for interrupts issued to the CSB host related to WDMA operation.

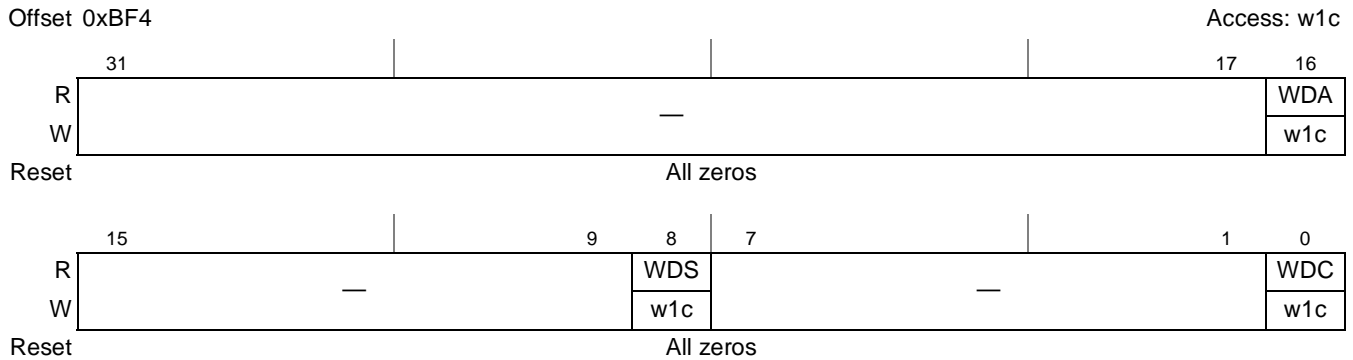


Figure 14-124. CSB System Write DMA Interrupt Status Register (PEX\_CSWDISR)

Table 14-122 defines the bit fields of PEX\_CSWDISR.

Table 14-122. PEX\_CSWDISR Register Fields Description

Bits	Name	Description
31–17	—	Reserved
16	WDA	WDMA transfer aborted. Indicates that a write DMA transaction was aborted.
15–9	—	Reserved
8	WDS	WDMA descriptor transfer completed. Indicates that a write DMA transaction corresponding to a descriptor successfully completed.
7–1	—	Reserved
0	WDC	WDMA chain descriptor transfer completed. Indicates that a write DMA transaction corresponding to the last descriptor in a chain successfully completed.

### 14.5.8.7 CSB System Read DMA Interrupt Status Register (PEX\_CSRDISR)

PEX\_CSRDISR, shown in Figure 14-125, maintains the status for interrupts issued to the CSB host related to the RDMA operation.

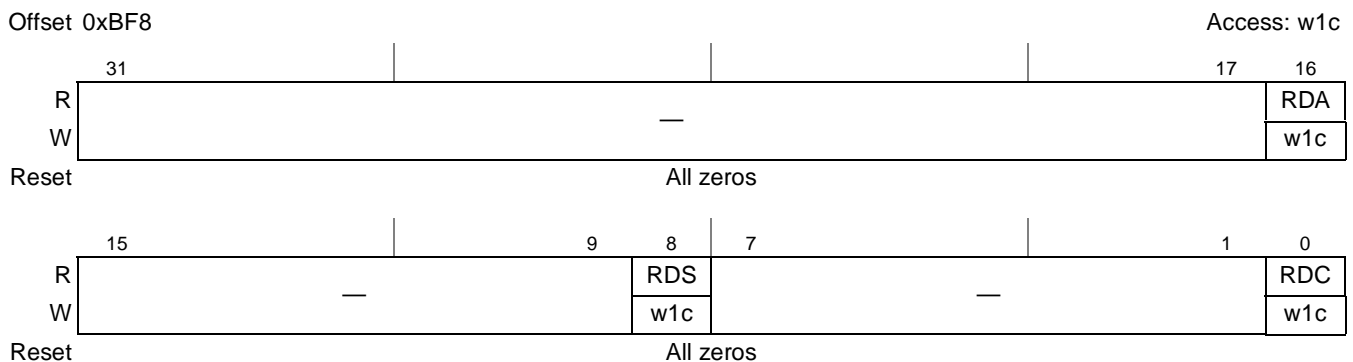


Figure 14-125. CSB System Read DMA Interrupt Status Register (PEX\_CSRDISR)

Table 14-123 defines the bit fields of PEX\_CSRDISR.

**Table 14-123. PEX\_CSRDISR Register Fields Description**

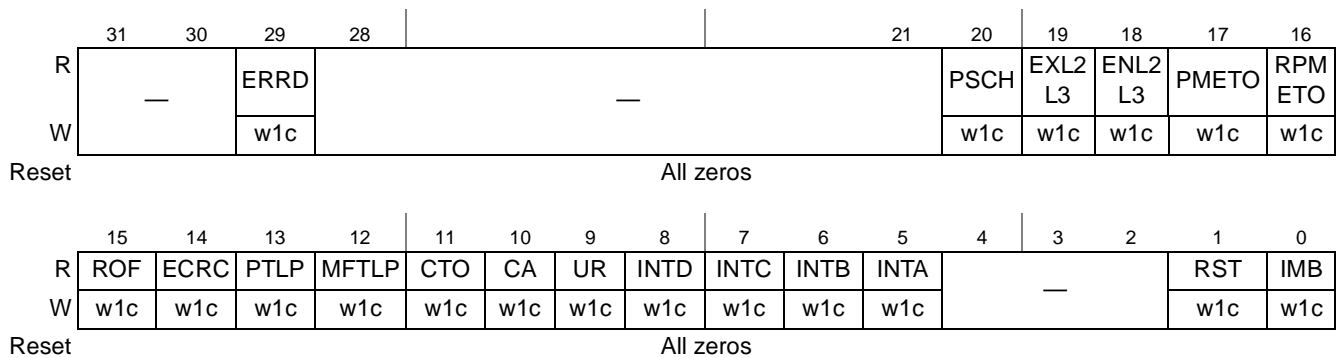
Bits	Name	Description
31–17	—	Reserved
16	RDA	RDMA transfer aborted. Indicates that a Read DMA transaction was aborted.
15–9	—	Reserved
8	RDS	RDMA descriptor transfer completed. Indicates that a read DMA transaction corresponding to a descriptor successfully completed.
7–1	—	Reserved
0	RDC	RDMA chain descriptor transfer completed. Indicates that a read DMA transaction corresponding to the last descriptor in a chain successfully completed.

### 14.5.8.8 CSB System Miscellaneous Interrupt Status Register (PEX\_CSMISR)

PEX\_CSMISR, shown in Figure 14-126, maintains the status for interrupt issued to the CSB.

Offset 0xBFC

Access: w1c



**Figure 14-126. CSB System Miscellaneous Interrupt Status Register (PEX\_CSMISR)**

Table 14-124 defines the bit fields of PEX\_CSMISR.

**Table 14-124. PEX\_CSMISR Register Fields Description**

Bits	Name	Description
31–30	—	Reserved
29	ERRD	Error detected. Indicates that a PCI Express event occurred. The PCI Express event is reported by the secondary status register (PCI Express secondary status register) at address 0x901E. Note that there is a secondary mask, the PCI Express interrupt mask register (PEX_SS_INTR_MASK), at address 0x95A0. Valid only for RC. This bit must be cleared after the interrupt is serviced and after the associated status registers in the PCI Express controller causing the interrupt are cleared.
28–21	—	Reserved

Table 14-124. PEX\_CSMISR Register Fields Description (continued)

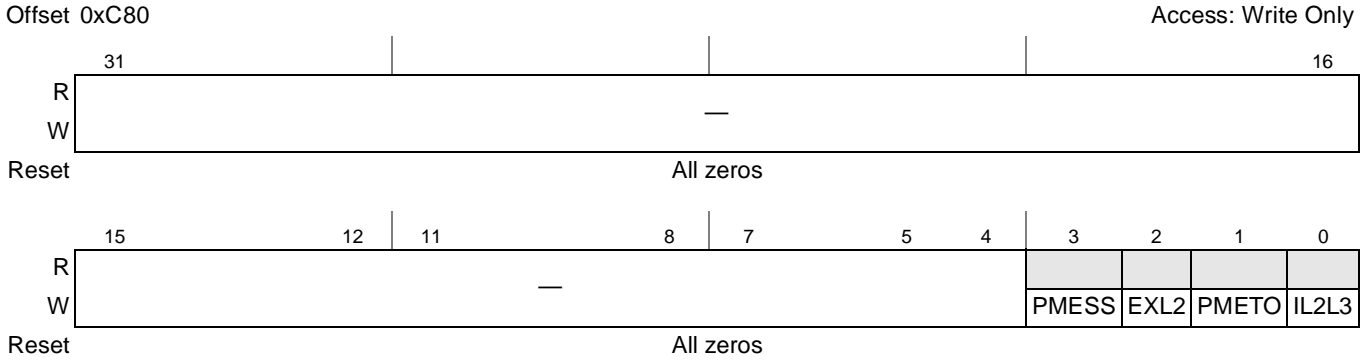
Bits	Name	Description
20	PSCH	Power state change. Indicates that a device power state change occurred. Change in Device power state (D state) for function-0. D-state can transition between the supported values of D0, D1, D2 and D3-hot. PM software changes D-state with a configuration write to PMCSR register in PM capability of the PCI Express. The new D-state is available in the corresponding field of the PMCSR register. This bit must be cleared after the interrupt is serviced.
19	EXL2L3	Exited L2/L3 state. Indication that the L2/L3 ready state has been exited and the current link state is L0. Traffic can be re-started on the link. This bit is set when the link switches from the L2/L3 ready to L0 state in response to an Exit L2 command. After issuing this command, the user must wait until the exited EXL2L3 bit is set before initiating traffic. This bit is valid only in RC mode. Setting this bit causes an interrupt to be sent to CSB side. The bit must be cleared after the interrupt is serviced.
18	ENL2L3	Entered L2/L3 state. Indication to the Power manager that it is safe to switch off power to the downstream device 100nsec after this bit is set. It is set when the PCI Express controller enters L2/L3 ready state. This bit is valid only in RC mode. Setting this bit causes an interrupt to be sent to the CSB side. The bit must be cleared after the interrupt is serviced.
17	PMETM O	PME Turn Off Ack timeout event. Indication to power manager software that it is safe to switch off power to the downstream device. It is set when the PCI Express controller detects that the timeout interval for receiving a PME_To_Ack message from the downstream device has expired. This bit is valid only in RC mode. This bit must be cleared after the interrupt is serviced.
16	RPMET O	Received PME Turn off message. Notifies that main power to the device is to be removed. After this notification is received, Uplink must not try to transmit TLPs or initiate PME requests because the power may be switched off. After this message is received, the user should indicate the readiness to lose power by setting the Initiate L2/L3 entry bit. This bit is valid only in EP mode. Setting this bit causes an interrupt to be sent to the CSB side. The bit must be cleared after the interrupt is serviced.
15	ROF	Receive overflow error. Indicates that the PCI Express reported a receive overflow error.
14	ECRC	ECRC error. Indicates that a TLP received by the PCI Express failed the ECRC check.
13	PTLP	Poisoned TLP. Indicates that a poisoned TLP was received by the PCI Express.
12	MFTLP	Malformed TLP. Indicates that a malformed TLP was received by the PCI Express
11	CTO	Completion timeout. Indicates that a PCI Express completion timeout occurred.
10	CA	Completer abort. Indicates that a PCI Express completion Abort was received.
9	UR	Unsupported request. Indicates that an unsupported request was received.
8	INTD	PCI Express INTD. Indicates that an INTD interrupt was received on the PCI Express link. Valid for RC applications only.
7	INTC	PCI Express INTC. Indicates that an INTC interrupt was received on the PCI Express link. Valid for RC applications only.
6	INTB	PCI Express INTB. Indicates that an INTB interrupt was received on the PCI Express link. Valid for RC applications only.
5	INTA	PCI Express INTA. Indicates that an INTA interrupt was received on the PCI Express link. Valid for RC applications only.
4–2	—	Reserved
1	RST	PCI Express reset. Indicates that a PCI Express reset occurred.
0	IMB	Inbound mailbox ready. Indicates that the inbound mailbox control register ready bit (PEX_IMBCR[READY]) was set and the CSB host can read the mailbox data.

## 14.5.9 PCI Express Power Management Registers

This section describes the PCI Express power management control register.

### 14.5.9.1 PCI Express Power Management Control Register (PEX\_PM\_CTRL)

This PCI Express PM Control Register shown in [Figure 14-127](#), is used to control the link power management by the PCI Express controller.



**Figure 14-127. PCI Express PM Control Register (PEX\_PM\_CTRL)**

[Table 14-125](#) defines the bit fields of the PEX\_PM\_CTRL.

**Table 14-125. PEX\_PM\_CTRL Register Fields Description**

Bits	Name	Description
31–4	—	Reserved. Must be zeros.
3	PMESS	PME Status Set. Set the PME Status bit in the PCI Express Power Management Status and Control Register of the configuration space (Offset 0x048). In EP mode, this also causes the PCI Express controller to send PM_PME message. PME message transmission is not supported in RC mode, but the PME status bit can still be set. PM PME message can be used to request for a device power state change. The message will be transmitted only if PME is enabled and if PME Turn off message has not been received by the endpoint. This field is Write only. Read always returns zero.
2	EXL2	Exit L2. This bit is valid only in RC mode and instructs the PCI Express controller to exit from L2/L3 ready state and move to L0 active state so that traffic can be re-started on the PCI-Express link. This bit can be set under the following conditions: the downstream device was shut down by the power manager (through L2/L3 protocol) and later has its power restored, whereas the upstream device (CI Express controller as RC) was in L2/L3 ready state (with power and clocks available). This field is Write only. Read always returns zero.



**Table 14-125. PEX\_PM\_CTRL Register Fields Description (continued)**

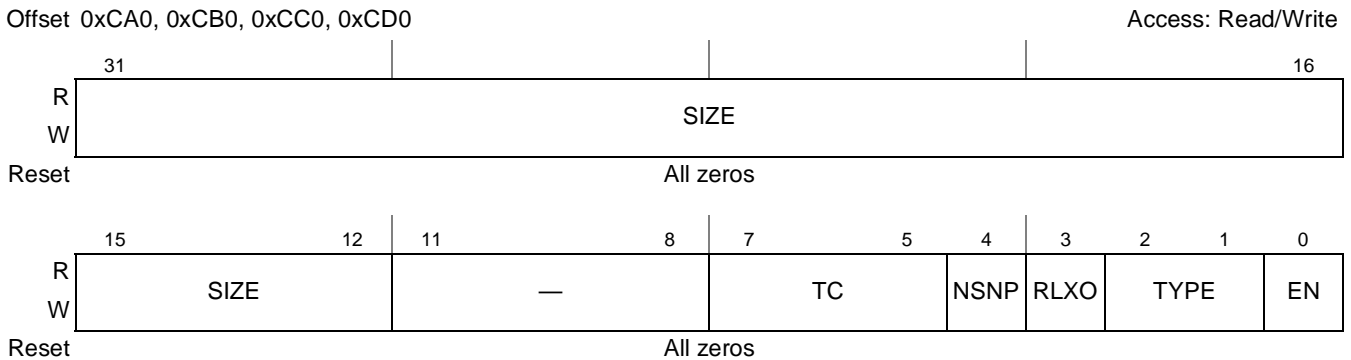
Bits	Name	Description
1	PMETO	Send PME Turn off message. This bit is valid only in RC mode and instructs the PCI Express controller to send PME_Turn_Off message to downstream devices. After setting this bit, the user must not try to transmit TLPs or initiate PME requests as the power may be switched off. This field is Write only. Read always returns zero.
0	IL2L3	Initiate L2/L3 entry. This bit is valid only in EP mode and instructs the PCI Express controller to transition to L2/L3 ready state. This bit has to be asserted only after preparing for power removal. After setting this bit, the user must not try to transmit TLPs or initiate PME requests as the power may be switched off. This field is Write only. Read always returns zero.

### 14.5.10 PCI Express Outbound Address Mapping Registers

The registers discussed in this section control the outbound transactions attributes and address mapping from the CSB domain to the PCI Express domain. These registers are used in both RC and EP modes, and serve both PIO and DMA transactions.

#### 14.5.10.1 PCI Express Outbound Window Attributes Register *n* (PEX\_OWAR0–PEX\_OWAR3)

PEX\_OWAR0–PEX\_OWAR3, shown in Figure 14-128, sets the attributes for the respective address window defined by the base and translation address registers for mapping of addresses related to CSB outbound transactions to PCI Express addresses.



**Figure 14-128. PCI Express Outbound Window Attributes Register *n* (PEX\_OWAR0–PEX\_OWAR3)**

Table 14-126 defines the bit fields of the PEX\_OWAR0–PEX\_OWAR3.

**Table 14-126. PEX\_OWAR0–PEX\_OWAR3 Register Fields Description**

Bits	Name	Description
31–12	SIZE	CSB window size. Indicates the size of window in bytes. The actual size is a concatenation of the SIZE field as most significant bits and 12 zeroes as least significant bits {SIZE[31–12], 0x000}.
11–8	—	Reserved. Must be zeros.

**Table 14-126. PEX\_OWAR0–PEX\_OWAR3 Register Fields Description (continued)**

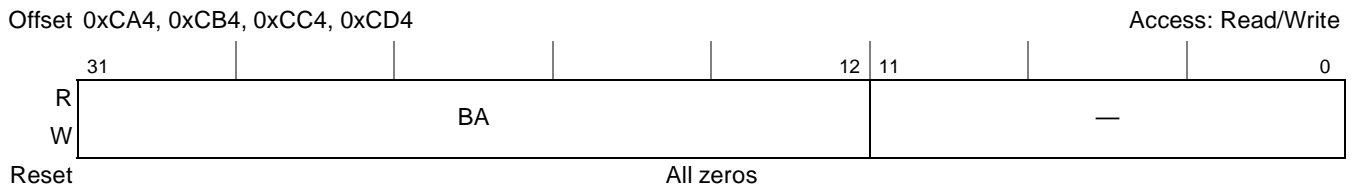
Bits	Name	Description
7–5	TC	Traffic class. Indicates the traffic class of the packet. Applicable only if user wants to send traffic using multiple TC but single VC.
4	NSNP	No snoop enable. When this bit and the PCI Express device control register [Enable No Snoop] bit are set, the No Snoop bit for the packet is enabled. This attribute is not applicable and must be cleared for configuration requests, I/O requests, and memory requests that are Message Signaled Interrupts. 0 PCI Express TLP snoop enabled 1 PCI Express TLP snoop disabled
3	RLXO	Relax ordering enable. When this bit and the PCI Express device control register [Enable Relaxed] bit are set, this bit enables the relaxed ordering bit for the packet. This applies only to memory transactions.
2–1	TYPE	Window type. Indicates the type to which CSB transactions to the window address are mapped. 00 CFG 01 I/O 10 Memory 11 Reserved
0	EN	Enable. Must be set to enable this window.

**NOTE**

The access destination of the configuration access is decided in the address converted with PEX\_OWTAR $n$  if the TYPE field in PEX\_OWAR $n$  is configured for CFG[00].

**14.5.10.2 PCI Express Outbound Window Base Address Register  $n$  (PEX\_OWBAR0–PEX\_OWBAR3)**

PEX\_OWBAR0–PEX\_OWBAR3, shown in [Figure 14-129](#), contains the base address of the CSB window for mapping to a PCI Express address. Note that the CSB base address must be aligned to 1 Kbyte. Inbound and outbound windows for the same bus should not overlap. Therefore, situations where an inbound window translation points back into an outbound window, or where an outbound translation window points back into an inbound window, are not allowed.



**Figure 14-129. PCI Express Outbound Window Base Address Register  $n$  (PEX\_OWBAR0–PEX\_OWBAR3)**

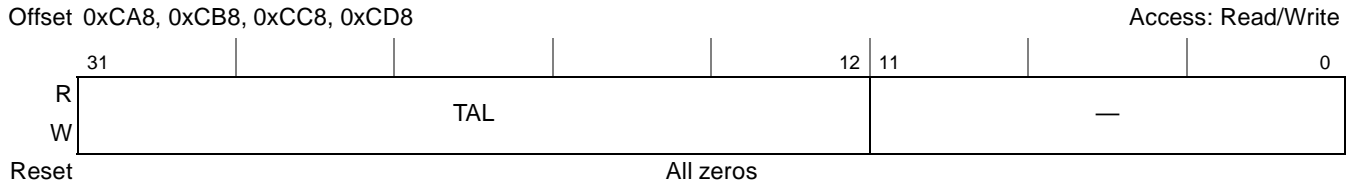
Table 14-127 defines the bit fields of the PEX\_OWBAR0–PEX\_OWBAR3.

**Table 14-127. PEX\_OWBAR $n$  Register Fields Description**

Bits	Name	Description
31–12	BA	Base address. The CSB window base address. Represents the CSB-based address for the window. The actual address is a concatenation of the BAR field as most significant bits and 12 zeroes as least significant bits {BA[31–12], 0x000}.
11–0	—	Reserved. Must be zeros.

### 14.5.10.3 PCI Express Outbound Window Translation Address Register Low $n$ (PEX\_OWTLARL0–PEX\_OWTLARL3)

PEX\_OWTLARL0–PEX\_OWTLARL3, shown in Figure 14-130, contain the lower base address of the PCI Express domain corresponding to this window. When this window is enabled and a CSB-based transaction hits its base address register, the address is translated to a PCI Express address, according to the PEX\_OWTLARL $n$  and PEX\_OWTLARH $n$  registers.



**Figure 14-130. PCI Express Outbound Window Translation Address Register Low  $n$  (PEX\_OWTLARL0–PEX\_OWTLARL3)**

Table 14-128 defines the bit fields of PEX\_OWTLARL $n$ .

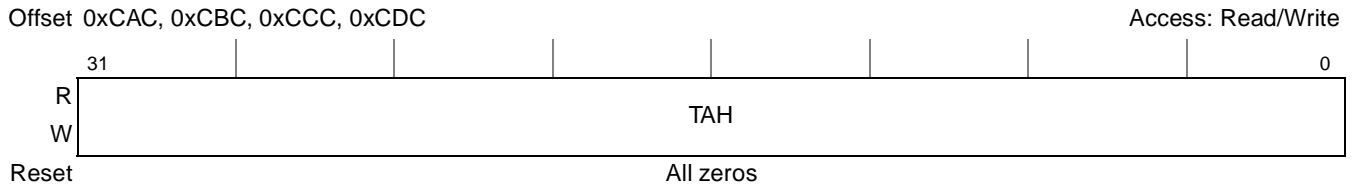
**Table 14-128. PEX\_OWTLARL $n$  Register Fields Description**

Bits	Name	Description
31–12	TAL	Translation address low. The lower portion of the PCI Express address base. The actual address is a concatenation of the TA field as most significant bits and 12 zeroes as least significant bits {TAL[31–12], 0x000}. The complete 64 bits address on the PCI Express bus is built of {PEX_OWTLARH[TALH], PEX_OWTLARL[TAL], 0x000}.
11–0	—	Reserved.

### 14.5.10.4 PCI Express Outbound Window Translation Address Register High $n$ (PEX\_OWTLARH0–PEX\_OWTLARH3)

PEX\_OWTLARH0–PEX\_OWTLARH3, shown in Figure 14-131, contains the higher base address of the PCI Express domain corresponding to this window. When this window is enabled and a CSB based transaction hits its base address register, the address is translated to a PCI Express address according to the

PEX\_OWTARL $n$  and PEX\_OWTARH $n$  registers. This register should be used in 64-bit addressing. Otherwise it should contain all zeroes.



**Figure 14-131. PCI Express Outbound Window Translation Address Register High  $n$  (PEX\_OWTARH0–PEX\_OWTARH3)**

Table 14-129 defines the bit fields of PEX\_OWTARH $n$ .

**Table 14-129. PEX\_OWTARH $n$  Register Fields Description**

Bits	Name	Description
31–0	TAH	Translation address high. Higher portion of the PCI Express address base ([63–32]). The complete 64-bit address on the PCI Express bus is built of {PEX_OWTARH[TAH],PEX_OWTARL[TAL], 0x000}.

### 14.5.11 PCI Express EP Inbound Address Translation Registers

The following registers are used as the base address for the CSB domain to translate the address of an inbound transaction from the PCI Express domain. They are valid only in End Point (EP) mode and operate in conjunction with the base address registers in the PCI Express configuration space.

When a PCI Express inbound transaction hits a valid address window defined by the PCI Express configuration space base address registers and the respective translation register is enabled, the incoming address is translated to a CSB domain address and the transaction is forwarded to the CSB.

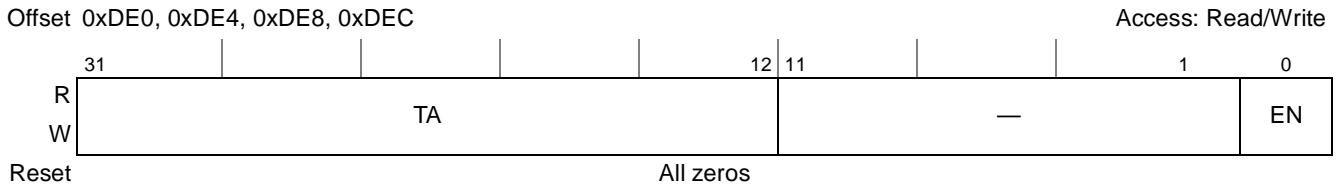
The correspondence between the PCI Express configuration space base address registers and the respective translation address registers is shown in Table 14-130.

**Table 14-130. EP Inbound Base and Translation Address Registers Correspondence**

Base Address Register (Configuration Space)	BAR Name	Type	TAR Address	TAR Name
0x010	BAR0	Window 0, 32-bit address	0xDE0	PEX_EPIWTAR0
0x014	BAR1	Window 1, 32-bit address	0xDE4	PEX_EPIWTAR1
0x018	BAR2	Window 2, 64-bit address, low portion	0xDE8	PEX_EPIWTAR2
0x01C	BAR3	Window 2, 64-bit address, high portion		
0x020	BAR4	Window 3, 64-bit address, low portion	0xDEC	PEX_EPIWTAR3
0x024	BAR5	Window 3, 64-bit address, high portion		

### 14.5.11.1 PCI Express EP Inbound Window Translation Address Register *n* (PEX\_EPIWTAR0–PEX\_EPIWTAR3)

PEX\_EPIWTAR0–PEX\_EPIWTAR3, shown in Figure 14-132, contain the CSB address to be mapped for a PCI Express inbound transaction hitting the respective BAR window. Inbound and outbound windows for the same bus should not overlap. Therefore, situations where an inbound window translation points back into an outbound window, or where an outbound translation window points back into an inbound window, are not allowed.



**Figure 14-132. PCI Express EP Inbound Window Translation Address Register *n* (PEX\_EPIWTAR0–PEX\_EPIWTAR3)**

Table 14-131 defines the bit fields of PEX\_EPIWTAR $n$ .

**Table 14-131. PEX\_EPIWTAR $n$  Register Fields Description**

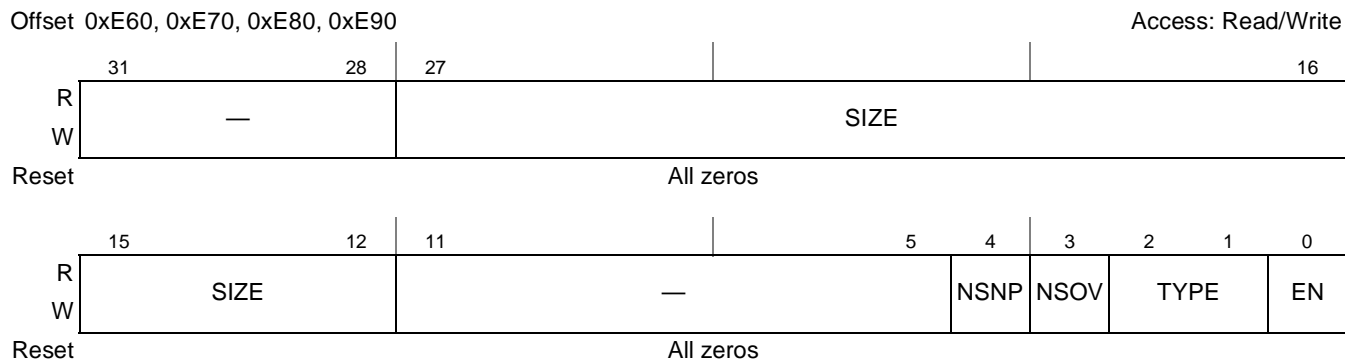
Bits	Name	Description
31–12	TA	Translation address. Contains the CSB base address to be mapped for a PCI Express inbound transaction hitting the respective BAR window. The actual address is a concatenation of the TA field as most significant bits and 12 zeroes as least significant bits {TA[31–12], 0x000}.
11–1	—	Reserved
0	EN	Enable. If set, indicates that the address mapping window is enabled.

### 14.5.12 PCI Express RC Inbound Address Mapping Registers

The registers discussed in this section control the inbound transactions attributes and address mapping from the PCI Express bus to the CSB domain. These registers are used only in RC mode, and they serve inbound transactions. When a PCI Express inbound transaction hits a valid address window defined by these registers and the respective translation register is enabled, the incoming address is translated to a CSB domain address and the transaction is forwarded to the CSB.

### 14.5.12.1 PCI Express RC Inbound Window Attributes Register *n* (PEX\_RCIWAR0 – PEX\_RCIWAR3)

PEX\_RCIWAR0–PEX\_RCIWAR3, shown in Figure 14-133, controls the mapping of a PCI Express inbound PIO transaction to a CSB transaction. This register is valid only in RC mode.



**Figure 14-133. PCI Express RC Inbound Window Attributes Register *n* (PEX\_RCIWAR0–PEX\_RCIWAR3)**

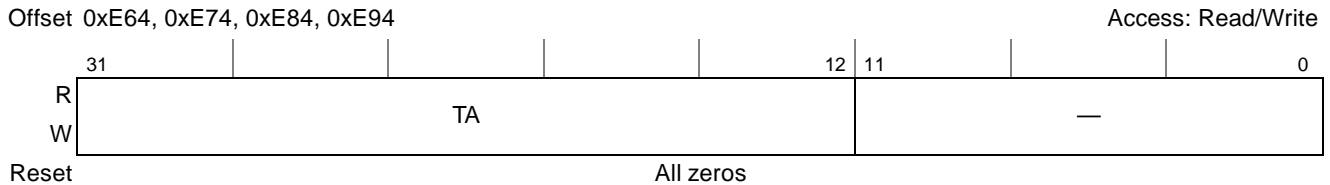
Table 14-132 defines the bit fields of PEX\_RCIWAR*n*.

**Table 14-132. PEX\_RCIWAR *n* Register Fields Description**

Bits	Name	Description
31–28	—	Reserved
27–12	SIZE	PCI Express window size. Indicates the size of the window in bytes. The actual size is a concatenation of the SIZE field as most significant bits and 12 zeroes as least significant bits {SIZE[27–12], 0x000}.
11–5	—	Reserved. Must be zeros.
4	NSNP	No snoop. If the no-snoop override enable bit in this register (NSOV) is set, this bit defines the snooping behavior on the CSB domain for the inbound packet hitting this window. This applies only to memory transactions. 0 CSB snoop enabled 1 CSB snoop disabled
3	NSOV	No-snoop override. If set, the No-Snoop attribute in the packet is overridden by the No Snoop bit in this register (NSNP). Otherwise, the No-Snoop bit in the inbound packet defines the snooping behavior. 0 Snoop behavior is defined by the inbound packet 1 Snoop behavior is defined by the NSNP field
2–1	TYPE	Type. Indicates the type of the window to which the PCI Express transactions are mapped. 00 Reserved 01 Reserved 10 Prefetchable memory. Inbound read transactions are optimized for CSB performance. The address and size of the actual memory read transaction may differ from those of the original PCI Express packet, aligning the first and last segments of the data read from the memory to cache line boundaries. 11 Non-prefetchable memory. Inbound read transactions from the PCI Express bus access the exact address and size of the memory. This mode is not optimized for CSB performance.
0	EN	Enable. Must be set to enable this window.

### 14.5.12.2 PCI Express RC Inbound Window Translation Address Register $n$ (PEX\_RCIWTAR0–PEX\_RCIWTAR3)

PEX\_RCIWTAR0–PEX\_RCIWTAR3), shown in Figure 14-134, contain the CSB address to be mapped for a PCI Express inbound transaction hitting the respective base address register of this window. These registers are valid only in RC mode. Inbound and outbound windows for the same bus should not overlap. Therefore, situations where an inbound window translation points back into an outbound window, or where an outbound translation window points back into an inbound window, are not allowed.



**Figure 14-134. PCI Express RC Inbound Window Translation Address Register  $n$  (PEX\_RCIWTAR0–PEX\_RCIWTAR3)**

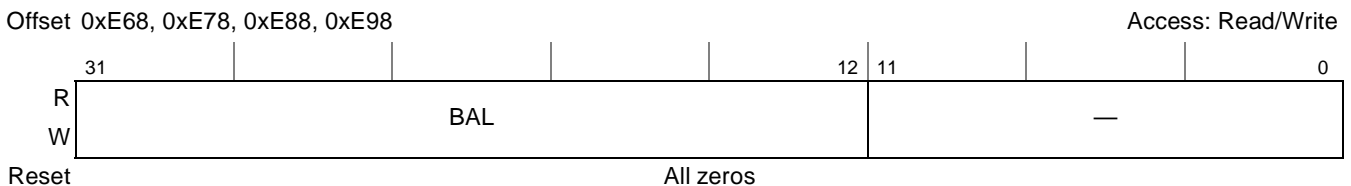
Table 14-133 defines the bit fields of PEX\_RCIWTAR $n$ .

**Table 14-133. PEX\_RCIWTAR $n$  Register Fields Description**

Bits	Name	Description
31–12	TA	Translation address. Contains the CSB base address to be mapped for a PCI Express inbound transaction hitting the respective base address register of this window. The actual address is a concatenation of the TA field as most significant bits and 12 zeroes as least significant bits {TA[31–12], 0x000}.
11–0	—	Reserved. Must be zeros.

### 14.5.12.3 PCI Express RC Inbound Window Base Address Register Low $n$ (PEX\_RCIWBARL0–PEX\_RCIWBARL3)

PEX\_RCIWBARL0–PEX\_RCIWBARL3, shown in Figure 14-135, contains the lower portion base address of the PCI Express domain corresponding to this window.



**Figure 14-135. PCI Express RC Inbound Window Base Address Register Low  $n$  (PEX\_RCIWBARL0–PEX\_RCIWBARL3)**

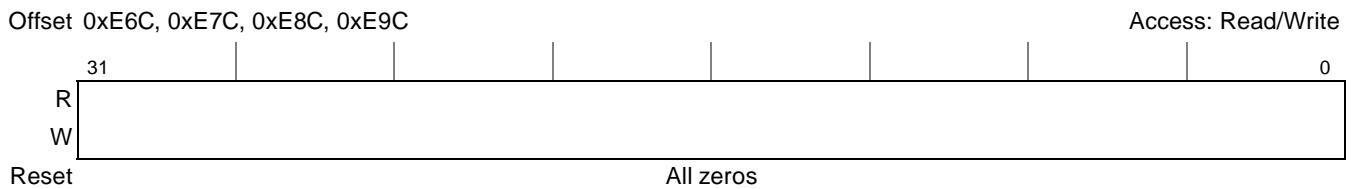
Table 14-134 defines the bit fields of PEX\_RCIWBARL<sub>n</sub>.

**Table 14-134. PEX\_RCIWBARL<sub>n</sub> Register Fields Description**

Bits	Name	Description
31–12	BAL	Base address low. Lower portion of the PCI Express address base. Represents the PCI Express-based address for the window. The actual address is a concatenation of the BAL field as most significant bits and 12 zeroes as least significant bits {BAL[31–12], 0x000}.
11–0	—	Reserved. Must be zeros.

### 14.5.12.4 PCI Express RC Inbound Window Base Address Register High *n* (PEX\_RCIWBARH0–PEX\_RCIWBARH3)

PEX\_RCIWBARH0–PEX\_RCIWBARH3, shown in Figure 14-136, contain the higher-portion base address of the PCI Express domain corresponding to this window. This register should be used in 64-bit addressing. Otherwise, it should contain all zeroes.



**Figure 14-136. CI Express RC Inbound Window Base Address Register High *n* (PEX\_RCIWBARH0–PEX\_RCIWBARH3)**

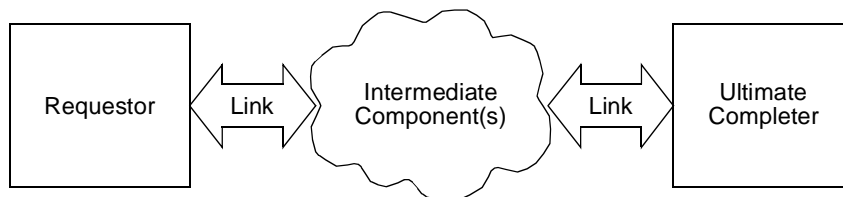
Table 14-135 defines the bit fields of PEX\_RCIWBARH<sub>n</sub>.

**Table 14-135. PEX\_RCIWBARH<sub>n</sub> Register Fields Description**

Bits	Name	Description
31–0	BAH	Base address high. Higher portion of the PCI Express address base ([63–32]). The complete 64-bit address on the PCI Express bus is built of {PEX_RCIWBARH[BAH],PEX_RCIWBARL[BAL], 0b0000000000}.

## 14.6 Functional Description

The PCI Express protocol relies on a requestor/completer relationship in which one device requests that a target device perform an action, 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 14-137. 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 14-138.

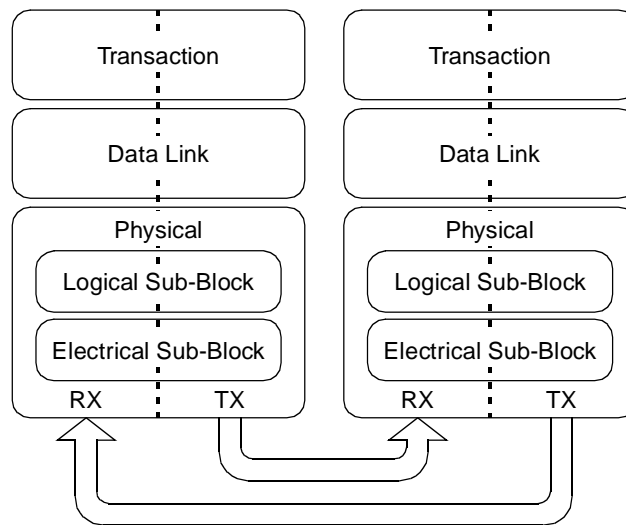


Figure 14-138. PCI Express High-Level Layering

Packets are formed in the transaction layer (TL) and data link layer (DLL), and each subsequent layer adds the necessary encoding and framing. 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.

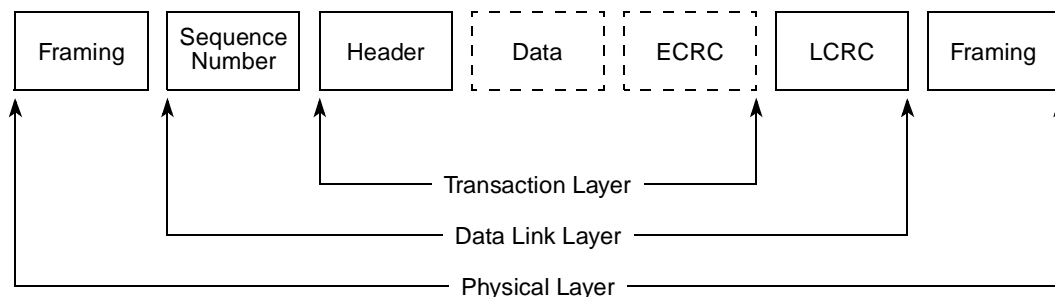


Figure 14-139. PCI Express Packet Flow

## 14.6.1 Architecture

This section contains the following:

- Section 14.6.1.1, “Address Translation Windows (ATMUs)”
- Section 14.6.1.2, “PCI Express Transactions”
- Section 14.6.1.3, “Byte Swapping”
- Section 14.6.1.4, “Outbound Byte Swapping”
- Section 14.6.1.5, “Inbound Byte Swapping”
- Section 14.6.1.6, “Transaction Ordering Rule”
- Section 14.6.1.7, “Memory Space Addressing”

- [Section 14.6.1.8, “I/O Space Addressing”](#)
- [Section 14.6.1.9, “Configuration Space Access”](#)
- [Section 14.6.1.10, “Inbound Messages”](#)

### 14.6.1.1 Address Translation Windows (ATMUs)

The device includes four general purpose inbound and outbound address translation windows which are used to map between the PCI Express domain and the CSB domain (referred as local address space). The outbound windows are common for both Root Complex and End Point modes, and their configuration registers reside in the CSB bridge space. For inbound windows there is a different set of configuration registers between Root Complex and End Point modes. EP inbound base address registers (BARs) reside in the standard configuration header space of the PCI Express core, while RC inbound base address registers (BARs) reside in the CSB bridge space. Because the CSB domain supports only 32 bit addressing, outbound base address registers and inbound translation registers are 32 bit only. [Table 14-136](#) specifies the available combinations between base address registers and translation address registers.

**Table 14-136. Address Translation Window Combinations**

Window Number	Type	BAR Name	BAR Address	TAR Name	TAR Address
<b>Outbound Windows—Common for Root Complex and End Point Modes</b>					
0	64-bit address, low portion	PEX_OWBAR0	0x9CA4	PEX_OWTARL0	0x9CA8
	64-bit address, high portion			PEX_OWTARH0	0x9CAC
1	64-bit address, low portion	PEX_OWBAR1	0x9CB4	PEX_OWTARL1	0x9CB8
	64-bit address, high portion			PEX_OWTARH1	0x9CBC
2	64-bit address, low portion	PEX_OWBAR2	0x9CC4	PEX_OWTARL2	0x9CC8
	64-bit address, high portion			PEX_OWTARH2	0x9CCC
3	64-bit address, low portion	PEX_OWBAR3	0x9CD4	PEX_OWTARL3	0x9CD8
	64-bit address, high portion			PEX_OWTARH3	0x9CDC
<b>Inbound Windows—End Point Mode</b>					
0	32-bit address	BAR0	0x010	PEX_EPIWTAR0	0xDE0
1	32-bit address	BAR1	0x014	PEX_EPIWTAR1	0xDE4
2	64-bit address, low portion	BAR2	0x018	PEX_EPIWTAR2	0xDE8
	64-bit address, high portion	BAR3	0x01C		
3	64-bit address, low portion	BAR4	0x020	PEX_EPIWTAR3	0xDEC
	64-bit address, high portion	BAR5	0x024		
<b>Inbound Windows—Root Complex Mode</b>					
0	64-bit address, low portion	PEX_RCIWBARL0	0x9E68	PEX_RCIWTAR0	0x9E64
	64-bit address, high portion	PEX_RCIWBARH0	0x9E6C		

Table 14-136. Address Translation Window Combinations

Window Number	Type	BAR Name	BAR Address	TAR Name	TAR Address
1	64-bit address, low portion	PEX_RCIWBARL1	0x9E78	PEX_RCIWTAR1	0x9E74
	64-bit address, high portion	PEX_RCIWBARH1	0x9E7C		
2	64-bit address, low portion	PEX_RCIWBARL2	0x9E88	PEX_RCIWTAR2	0x9E84
	64-bit address, high portion	PEX_RCIWBARH2	0x9E8C		
3	64-bit address, low portion	PEX_RCIWBARL3	0x9E98	PEX_RCIWTAR3	0x9E94
	64-bit address, high portion	PEX_RCIWBARH3	0x9E9C		

The attributes of the outbound windows are controlled by the PEX\_OWAR<sub>n</sub> registers and the attributes of the RC inbound windows are controlled by the PEX\_RCIWAR<sub>n</sub> registers, residing in the CSB bridge address space. The attributes of the EP inbound windows are controlled by both the translation registers, PEX\_EPIWTAR<sub>n</sub>, in the CSB bridge space, and by the PCI Express BAR configuration registers residing in the PCI Express core address space.

#### NOTE

For outbound transactions, both the PCI Express DMA and CSB Masters share the same set of windows.

### 14.6.1.2 PCI Express Transactions

Table 14-137 lists the PCI Express transactions supported by the device as an initiator and a target.

Table 14-137. PCI Express Transactions

PCI Express Transaction	MPC8378E/MPC8377E Support as an Initiator	MPC8378E/MPC8377E Support as a Target	Definition
Mrd	Yes	Yes	Memory Read Request
MRdLk	No	No	Memory Read Lock
MWr	Yes	Yes	Memory Write Request to memory-mapped PCI-Express space
IORd	Yes (RC only)	No	I/O Read request
IOWr	Yes (RC only)	No	I/O Write Request
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
CfgWr1	Yes (RC only)	No	Configuration Write Type 1
Msg	Yes	Yes	Message Request
MsgD	No	No	Message Request with data payload
Cpl	Yes	Yes	Completion without Data

Table 14-137. PCI Express Transactions (continued)

PCI Express Transaction	MPC8378E/MPC8377E Support as an Initiator	MPC8378E/MPC8377E Support as a Target	Definition
CplD	Yes	Yes	Completion with Data
CplLk	No	No	Completion for Locked Memory Read without Data
CplDLk	No	No	Completion for Locked Memory Read with Data

### 14.6.1.3 Byte Swapping

Byte swapping is executed when either an inbound transaction (that is, a write or response) to the CSB or an outbound transaction with data (that is, a write) to PCI Express is performed.

### 14.6.1.4 Outbound Byte Swapping

Byte swapping is executed when an outbound transaction with data (that is, a write) to PCI Express is performed. Figure 14-140 shows the outbound data swapping.

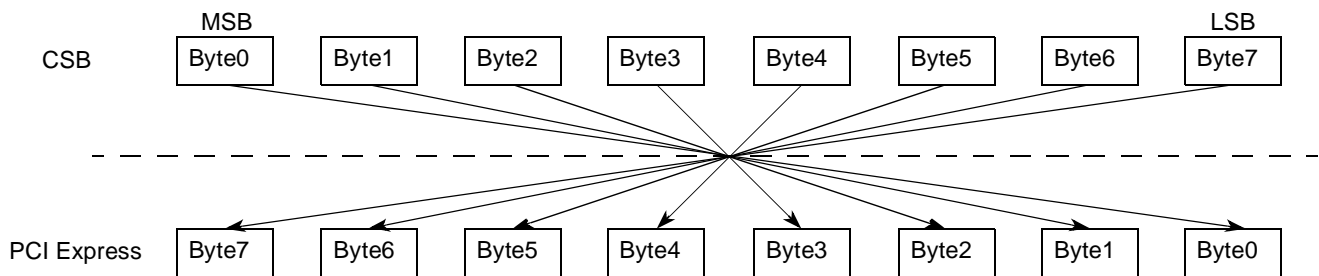


Figure 14-140. Outbound Byte Swapping

### 14.6.1.5 Inbound Byte Swapping

Byte swapping is executed when an inbound transaction with data (that is, a write or response) to the CSB is performed.

### 14.6.1.6 Transaction Ordering Rule

In general, transactions are serviced in the order that they were received. However, transactions can be reordered as they are sent due to stalled conditions such as the internal buffer full condition. Below are the ordering rules for sending the next outstanding request.

- A posted request can and will bypass all other transactions except another posted request.
- Completion can and will bypass non-posted and posted requests only if the relaxed ordering (RO) bit of the PCI Express packet's header is set.
- A non-posted request cannot bypass a posted or non-posted request but can bypass completion if the relaxed ordering bit is set.

Note that it is possible for one outbound configuration transaction to bypass another outbound configuration transaction due to CRS status and the ability for hardware to retry the transaction.

#### 14.6.1.7 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 header for a 32-bit address packet is 0 while a 64-bit address packet has a 1 indication. A memory read transaction has settings of 00000 for the Type[4–0] field in the PCI Express header and 0 for Fmt[1]. A memory write transaction has the Type[4–0] field in the PCI Express header as 00000 and Fmt[1] as 1. As an initiator, the controller is capable of sending a 32- or 64-bit memory packet depending on the window translation address. Any transaction from the CSB that has a translated address greater than 4G after going through the translation window is sent as a 64-bit memory packet. Otherwise, a 32-bit memory packet is sent. As a target device, the controller can decode a 32- or 64-bit memory packet using two 32-bit inbound windows and two 64-bit inbound windows. All inbound addresses are translated to the CSB address, which is 32 bits wide.

#### 14.6.1.8 I/O Space Addressing

The PCI Express controller does not support I/O transactions as a target. As an initiator, the controller can send I/O transactions in RC mode by programming one of the outbound translation window's attributes to send I/O transactions. All I/O transactions access only a 32-bit address I/O space. An I/O read transaction has the Type[4–0] field in the PCI Express header as 00010 and the Fmt[1] as 0. An I/O write transaction has the Type[4–0] field in the PCI Express header as 00010 and the Fmt[1] as 1.

#### 14.6.1.9 Configuration Space Access

To access the PCI Express controller's internal configuration header by MPC8308 itself, the only mechanism supported is the direct access by means of the CSB, since the whole internal configuration space is memory-mapped. This is true regardless the PCI Express controller is configured as RC or EP.

If the PCI Express controller is configured as RC,

- Inbound configuration transaction is not supported.
- Outbound configuration transaction to access downstream PCI Express devices is supported. The only mechanism can be used to initiate either Type 0 or Type 1 configuration cycle is via outbound ATMU windows.

If the PCI Express controller is configured as EP,

- Outbound configuration transaction is not supported. In other words, the PCI Express EP controller does not generate configuration transactions in EP mode.
- Inbound configuration transaction to access the PCI Express EP controller's configuration space is supported.

##### 14.6.1.9.1 Outbound ATMU Configuration Transaction Generation (RC)

In RC mode, the PCI Express controller can generate both Type 0 and Type 1 configuration cycles to access downstream PCI Express devices by means of the outbound ATMU windows mechanism.

As RC the PCI Express controller configuration access mechanism utilizes a memory-mapped address space to access device configuration registers. To achieve this, software can program the TYPE field of the PEX\_OWARN register to 0x0 in one of the outbound ATMU windows to perform a configuration access. The other bit fields of the PEX\_OWARN register should be programmed as below:

- TC = 0x0
- NSNP = 0
- RLXO = 0
- EN = 1

The SIZE field of the PEX\_OWARN register should be set to a value to correspond with the BA (base address) field of the base address register (PEX\_OWBARn), normally based on the Bus Number(s) of the downstream PCI Express device(s) to be accessed. The base address registers, PEX\_OWBARn, set the CSB address window for the configuration transactions. The translation address registers, PEX\_OWTARLn, can be used to define the translated PCI Express address of the CSB-based configuration transaction.

Once the PCI Express outbound window attributes register (PEX\_OWARN), base address register (PEX\_OWBARn) and translation address register (PEX\_OWTARLn) are fully defined, a CSB-based memory transaction hitting the defined base address register will be converted to an external PCI Express configuration transaction cycle appeared on the downstream link of the PCI Express RC controller. In this case, the CSB memory address determines the configuration register accessed and the memory data returns the contents of the addressed register. Software must only issue 4-byte or less access to the ATMU configuration window and the access cannot cross a 4-byte boundary.

The mapping from the CSB local address to PCI Express configuration space is defined by the Outbound ATMU as in a memory transaction. The actual PCI Express configuration header will be formatted from the CSB address and the ATMU translation, defined by PEX\_OWTARLn.

The formatted address defines the configuration transactions parameter, as shown in [Table 14-138](#). Note that there is no byte swapping for the address itself, although the programming of the registers content do require byte swapping. The table also translates between the bit ordering commonly used by the PowerPC terminology (0–31) and the bit ordering used by the PCI Express terminology (31–0).

**Table 14-138. Configuration Address Mapping**

CSB Address Bits Numbering	PCI Express Address Bits Numbering	PCI Express Configuration Space
0–7	31–24	Bus number
8–12	23–19	Device number
13–15	18–16	Function number
16–19	15–12	Reserved
20–23	11–8	Extended register number
24–29	7–2	Register number

Table 14-138. Configuration Address Mapping (continued)

CSB Address Bits Numbering	PCI Express Address Bits Numbering	PCI Express Configuration Space
30–31	1–0	Reserved

**Note:** It is the user's responsibility to set the reserved bit fields to zero (especially bits 15–12).

**Note:** The configuration cycle generation mechanism does not differentiate from internal or external configuration cycle. This means that any transaction which hits a configuration window will be passed to the PCI Express link with a relevant transaction type.

### NOTE

The location of the configuration fields described in the table above are slightly different than the definition of a “flat memory addressing method” in the PCI Express specification, and are directly aligned to the fields in the configuration transaction header. The user software should take this difference into considerations.

The PCI Express RC controller initiates the Type 0 or Type 1 configuration cycle on its downstream link based on the following rules:

- If the bus number of the CSB-initiated transaction equals the secondary bus number from Type 1 header of RC's configuration space and the device number is 0, a Type 0 configuration cycle will be sent to the link.
- If the bus number of the CSB-initiated transaction does not equal the RC's primary bus number, and does not equal the secondary bus number (from RC's Type 1 header), and is less than or equal to the subordinate bus number (from RC's Type 1 header), a Type 1 configuration cycle will be sent to the link. Note that according to PCI and PCI Express base specifications, the relationship where the Secondary Bus Number  $\leq$  Subordinate Bus Number must be ensured when configuring the two bus numbers within RC's Type 1 header.
- For all other cases, the PCI Express RC controller will issue a configuration cycle on the link whenever an outbound configuration window is hit on the CSB side, no matter what the parameters are. It is the software driver's responsibility to block transactions with unsupported bus, device, and function numbers and return “1”s for such reads. If the bus number in the CSB-initiated transaction equals the primary bus number of RC hitting the outbound configuration window, software error will occur which must be handled by the driver.

The following is an example showing how to configure the related registers of one of the MPC8308 outbound windows for configuration transaction generation purpose. As MPC8308's default 8 Mbyte boot ROM location can be configured at either 0x0000\_0000 to 0x007F\_FFFF (at bottom 8-MByte local address) or 0xFF80\_0000 to 0xFFFF\_FFFF (at top 8 Mbyte local address), the software must ensure that the base address is configured correctly in the OWBARn such that the outbound window does not overlap with the configured boot ROM location. To simplify the illustration, this example uses the following assumptions:

- The Boot ROM location is configured to locate within 0x0000\_0000 to 0x007F\_FFFF.
- The overall PCI Express system has a total of 16 buses to be configured.

Sixteen buses mean that the bus number can range from 0x00 to 0x0F. In general, if there are  $2^n$  bus numbers to be configured,  $n$  number of address bits are needed to represent the variation of bus number ranging from 0 to  $2^n - 1$ . For this example, four address bits from CSB[4–7] are required to represent the bus number variation and therefore become the most significant four bits within the total 28 bit offset (CSB[4–31], including reserved bits) of the outbound window to be configured. The CSB[0–3] in this case will not participate in the bus number mapping process as shown in [Table 14-138](#), where all eight bits of CSB[0–7] are mapped to PCI Express address bits [31–24] to represent the possible of 256 bus numbers within a very large system. In other words, in the example, CSB[0–3] become the four most significant bits of the base address of the outbound window to be configured and will be translated to a new “address” in the PCI Express space as defined by the corresponding PEX\_OWTARLn.

Based on the above information, the most significant four bits of the base address of the outbound window came from CSB[0–3] must be unique within the total 4 Gbyte local CSB memory space. This essentially defines a window with size of 256 Mbytes, since the lower 28 bits are offset. For this example, assume a 256 Mbyte outbound window is therefore defined between 0x5000\_0000 and 0x5FFF\_FFFF in local CSB space. This yields the PEX\_OWBARn’s BA[31–12] to be 0x5000\_0, with the actual base address of the outbound window as 0x5000\_0000. The SIZE[31–12] field of the PEX\_OWARn register is 0x1000\_0 to reflect the actual size of this outbound window as 0x1000\_0000 or 256 Mbytes.

Note that the final configuration transaction to be generated is based on the information gathered from two areas: the most significant four bits from the defined TAL (translation address low) bit field of PEX\_OWTARLn and the lower order bits from the CSB[4–31] offsets directly mapped to PCI Express address bits [27–0]. As mention in the note section of [Table 14-138](#), software should ensure all the reserved bits within CSB[4–31] are filled with zeros.

Since TAL[31–24] (total of eight bits) of PEX\_OWTARLn could be used for mapping the bus number bit field for a general configuration transaction, while in this example the bus number to be configured ranges from 0x00 to 0x0F, the most significant four bits (TA[31–28]) are not needed for the bus number mapping and therefore must be configured as 0x0. The TA[27–12] bit field of PEX\_OWTARLn is left as all zeros. This yields the PEX\_OWTARLn’s TAL[31–12] to be 0x0000\_0, with the actual translation address of the outbound window as 0x0000\_0000. Since the size of this window is 256 Mbytes, the upper limit of the translation address is then locate at 0x0FFF\_FFFF.

With the outbound window configured as above, if software intends to scan the PCI Express bus and attempts to probe the first bus immediately underneath the PCI Express RC, the software will need to issue a Configuration Transaction to read Register Number 0 (Vendor ID Register) at Bus Number/Device Number/Function Number of 1/0/0, assuming the RC’s secondary bus number has been configured as 0x1 along with subordinate bus number being configured with a big number like 0xFF initially. As long as the LAW is configured to ensure that the local address space between 0x5000\_0000 and 0x5FFF\_FFFF is configured for PCI Express, a CSB-based memory read transaction to local address 0x5100\_0000 initiated by software will be translated to a transaction hitting PCI Express RC controller with PCI Express address of 0x0100\_0000. Upon receiving this CSB-based transaction, the RC controller checks the Type attribute of this outbound window and realizes the transaction is of a configuration type, instead of directly using the translated address as in usual memory transaction, the RC controller starts compose a configuration transaction with header information from this translation address based on the mapping defined in [Table 14-138](#). The decode of the mapping process yields a type 0 configuration transaction to be generated



on the PCI Express link with Bus Number = 1, Device Number = 0, Function Number = 0, and Register Number = 0.

Similarly, if the software intends to read the above EP's configuration space offset 0x440, it can generate a CSB-based memory transaction to address 0x5100\_0440. Once the transaction hitting the above configured outbound window, the ATMU translates it into a transaction with PCI Express address 0x0100\_0440. This RC controller, once receiving the transaction, will issue a Type 0 configuration transaction on the PCI Express link with Bus Number = 1, Device Number = 0, Function Number = 0, Extended Register Number = 0x4, and Register Number = 0x40.

#### NOTE

In the example above the translation address register (PEX\_OWTARL $n$ ) is set once. It is also possible to use a dynamic approach of updating the translation address register before every configuration access. In this method the PEX\_OWBAR $n$  and the PEX\_OWAR $n$  for the configuration window can be set for a relatively small address range, but the software needs to adjust the translation address register (PEX\_OWTAR $n$ ) for the configuration window to the desired parameters prior to issuing the configuration transaction.

The programming of the ATMU registers must guarantee that there is no overlap between address bits defined by the base address and size of the window, and address bits defined by the translation address. In other words, the bits in the lower portion of the PEX\_OWTARL $n$  covered by the base address must be zero.

#### 14.6.1.9.2 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 up to 4096 bytes of the PCI Express configuration area. While in EP mode, the PCI Express controller does not support generating configuration accesses as a master. 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 will get a response with an error and can lead to unexpected results.

#### 14.6.1.10 Inbound Messages

The following tables lists the messages and the actions that take place depending on whether RC or EP mode is configured. The actual events are logged in the PCI Express Root Error Status Register and in the CSB System Miscellaneous Interrupt Status Register (PEX\_CSMISR). See [Section 14.4.5.10, "PCI Express Root Error Status Register](#), [Section 14.5.8.4, "CSB System Miscellaneous Interrupt Enable Register \(PEX\\_CSMIER\)](#) and [Section 14.5.8.8, "CSB System Miscellaneous Interrupt Status Register \(PEX\\_CSMISR\)](#) for further details.

Table 14-139 lists the messages and the actions that take place in RC mode.

**Table 14-139. PCI Express RC Inbound Message Handling**

Name	Code[7:0]	Routing[2:0]	Action
Assert_INTA	0010 0000	100	Set INTA event
Assert_INTB	0010 0001	100	Set INTB event
Assert_INTC	0010 0010	100	Set INTC event
Assert_INTD	0010 0011	100	Set INTD event
Deassert_INTA	0010 0100	100	Clear INTA event
Deassert_INTB	0010 0101	100	Clear INTB event
Deassert_INTC	0010 0110	100	Clear INTC event
Deassert_INTD	0010 0111	100	Clear INTD event
PM_Active_State_Nak	0001 0100	100	No action taken
PM_PME	0001 1000	000	Set PM_PME event
PME_Turn_Off	0001 1001	011	No action taken
PME_TO_Ack	0001 1011	101	Log entered_I23_state in PME and message detect register and generate interrupt to IPIC if enabled
ERR_COR	0011 0000	000	Set correctable error event
ERR_NONFATAL	0011 0001	000	Set non-fatal error event
ERR_FATAL	0011 0011	000	Set fatal error event
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	No action taken

Table 14-140 lists the messages and the actions that take place in EP mode.

**Table 14-140. 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	1. Log in PME and message detect register if enabled. 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	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	No action taken

## 14.6.2 Interrupts

Message signaled interrupt (MSI) generation and handling are supported; however there are subtle differences depending on whether the device is configured as an RC or EP device.

## 14.6.2.1 EP Interrupt Generation

Hardware MSI generation is supported for the interrupt events described in [Section 14.5.7, “PCI Express Host Interrupt Registers.”](#)

### 14.6.2.1.1 Hardware MSI Generation

Host software must set up the MSI capability registers to enable MSI mode and put the correct MSI address and data values into the MSI capability registers prior to setting up various interrupt event enable bits in the PEX\_HIER register to enable the generation of the correct MSI cycle to RC.

Note that the value being programmed by the host software into the MSI Data register of EP’s Type 0 configuration space is a 16-bit base message data pattern, which is referred to as “base vector [15–0]” in the description below:

- If only one MSI message is desired, the EP software can directly use this base vector as the interrupt vector for MSI interrupt generation. In such case, there is no need to program any of the vector registers among PEX\_HOPIVR, PEX\_HIPIVR, PEX\_HWDIVR, PEX\_HRDIVR, and PEX\_HMIVR. The PEX\_HIER register setting determines which event can trigger this single MSI interrupt message to RC. Note that multiple interrupt events are allowed to share the same MSI interrupt vector.
- If multiple MSI messages are desired, the multiple message capable bit field of EP’s MSI message control register can be used to indicate how many MSI messages (in the power of two, up to 32 messages allowed per EP) the EP wants to use. During configuration stage, after examining the above desired value, the Host software will allocate the actual number of MSI messages for an EP to use by configuring the multiple message enable bit field in the same register, in addition to programming the base vector in EP’s MSI data register. Once this is accomplished, the EP software can program the IVEC bit field of each individual vector register based on the number of MSI messages allocated by host. The IVEC value must be unique for the same EP and start from 0x00. At last, the EP software can set the corresponding interrupt event enable bits in the PEX\_HIER to enable the Hardware MSI generation. The actual MSI data value for a given interrupt event used by the EP in its MSI message to RC is formed by the concatenation of the base vector [15–5] and the IVEC [4–0] value of the corresponding interrupt event.

As an example, if the value of the multiple message enable bit field allocated by host software is 010b (4 MSI messages allocated) and the base vector being programmed by host software in EP’s MSI Data Register is 0x55A0 (lower-16 bits little endian), the actual MSI data values of all possible MSI messages can be used by the EP are 0x0000\_55A0, 0x0000\_55A1, 0x0000\_55A2, and 0x0000\_55A3. The EP software only needs to program the four possible lower-order bits (0x00, 0x01, 0x02, and 0x03) as unique IVEC values in its vector registers among PEX\_HOPIVR, PEX\_HIPIVR, PEX\_HWDIVR, PEX\_HRDIVR and PEX\_HMIVR. If both OPAIE and OPCIE bit fields are enabled in the PEX\_HIER register, assuming PEX\_HOPIVR register’s IVEC bit field is programmed as 0x03h, when any one of these two interrupt events occurs, the EP will use 0x0000\_55A3 as the actual MSI data value in its MSI message sent to RC. Once receiving such MSI message, the device driver running at RC is responsible to issue a read to EP’s PCI Express Interrupt Status Register (PEX\_HISR) to find out exactly which of the two interrupt events caused the interrupt.

### 14.6.2.1.2 Software MSI Generation

Host software needs to set up the MSI capability registers to enable MSI mode and put the correct values for the MSI address and data register. Next, local software must 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 determines the number of allocated messages in the MSI capability register and allocates the appropriate data values to use. A write to the MSI ATMU window with the appropriate data value generates the MSI transaction to the RC.

### 14.6.2.2 RC Handling of MSI Interrupt

An MSI interrupt cycle must hit into the IMMRBAR window (window 0) with the address offset that points to MIISR register in the IPIC. Note that the host software must configure the EP's MSI capability register so that an MSI cycle generated from the device is routed to the correct MIISR register in the IPIC and for the appropriate interrupt to be generated to the core.

### 14.6.2.3 Initial Credit Advertisement

To prevent overflowing of the link partner's receiver buffers and for compliance with ordering rules, the transmitter cannot send transactions unless it has enough credits to send. Each device maintains a flow control (FC) credit pool. The FC information is conveyed between two links by DLLPs during link training (initial credit advertisement). The transaction layer performs the FC accounting functions. It functions as the FC gate. One unit of FC is 4 DWs (16 bytes) of data.

**Table 14-141. Initial Credit Advertisement**

Credit Type	Initial Credit Advertisement
PH (memory write, message write)	4
PD (memory write, message write)	$(256 \div 16) \times 4 = 64$
NPH (memory read, I/O read, cfg read)	8
NPD (I/O write, cfg write)	2
CPLH (memory read completion, I/O R/W completion, cfg R/W completion)	Infinite
CPLD (memory read completion, I/O read completion, cfg read completion)	Infinite

## 14.6.3 Mailbox

The mailbox mechanism is useful when the device is in EP mode, and enables exchanging information between the remote PCI Express root complex and the local host using interrupts and data storage registers. There are sets of register for both inbound and outbound mailbox messages and interrupts.

### 14.6.3.1 Outbound Mailbox

The EP local host uses the outbound mailbox messages to signal the remote RC device across the PCI Express link. The local host, for example the e300 core, stores the required message in the PCI Express outbound mailbox data register (PEX\_OMBDR) and then initiates an interrupt (MSI) to the remote PCI Express device. When the remote PCI Express RC device receives the interrupt it can perform

a memory read from the mailbox data register and read the message. The following steps are required in order to use the outbound mailbox mechanism.

1. The RC should set the MSIE bit of the PCI Express MSI message control register (address 0x72 of the configuration space) to enable the generation of MSI.
2. The EP local host should set the OMBIE bit of the PCI Express host interrupt enable register (PEX\_HIER) to enable interrupt generation to the PCI Express (MSI) at the event of setting the READY bit of the PCI Express outbound mailbox control register (PEX\_OMBCR).
3. The EP local host should program the IVEC field of the PCI Express host miscellaneous interrupt vector register (PEX\_HMIVR) with an appropriate vector value. This value, along with the value programmed in the EP's PCI Express MSI message control register's MME field determines the MSI message data sent to the RC. For example, if the MME field has a value of N, then the lower N bits of the MSI message data are replaced with the lower N bits of the PEX\_HMIVR register.
4. The EP local host should program the MBD field of the PCI Express outbound mailbox data register (PEX\_OMBDR) with the message to be read by the PCI Express remote device (user defined).
5. The EP local host should set the READY bit of the PCI Express outbound mailbox control register (PEX\_OMBCR). This will generate an interrupt (MSI) to the PCI Express root complex.
6. The PCI Express RC, after receiving the MSI will perform a memory read to the EP's PCI Express outbound mailbox data register (PEX\_OMBDR) and get the message content.
7. The PCI Express RC should perform a memory write to clear the READY bit of the EP's PCI Express outbound mailbox control register (PEX\_OMBCR).
8. The EP can repeat steps 3–5 with an appropriate MSI and message data after verifying that the PEX\_OMBCR[READY] is cleared.

### 14.6.3.2 Inbound Mailbox

The remote PCI Express RC device uses the inbound mailbox messages to signal the EP local host across the PCI Express link. The RC performs a memory write and stores the required message in the PCI Express inbound mailbox data register (PEX\_IMBDR) and then initiates an interrupt to the local host by performing a memory write and setting the READY bit of the PCI Express inbound mailbox control register (PEX\_IMBCR). When the local host detects the interrupt, it can read the message from the mailbox data register. The following steps are required in order to use the outbound mailbox mechanism.

1. The EP local host should enable PCI Express interrupts by programming the integrated programmable interrupt controller (IPIC).
2. The EP local host should set the IMBIE bit of the CSB system miscellaneous interrupt enable register (PEX\_CSMIER), to allow interrupt at the event of mailbox ready.
3. The PCI Express RC should perform a memory write and store the required message in the EP's PCI Express Inbound Mailbox Data Register (PEX\_IMBDR).
4. The PCI Express RC should perform a memory write and set the READY bit of the EP's PCI Express inbound mailbox control register (PEX\_IMBCR). This will issue an interrupt to the local host.

5. The local host can read the message content from the PCI Express inbound mailbox data register (PEX\_IMBDR).
6. The local host should clear PEX\_IMBCR[READY] and all the interrupt event and status bits in the relevant registers.
7. The RC can repeat steps 3–4 after verifying that the PEX\_IMBCR[READY] is cleared.

## 14.6.4 Power Management

All device power states are supported except D3cold. In addition, all link power states are supported except the L2 and L3 states. Only L0s ASPM (active state power management) mode is supported if enabled by configuring the link control register 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 14-142. Power Management State Supported**

Component D-State	Permissible Interconnect State	Action
D0	L0, L0s	In full operation.
D1	L1	All outbound traffic is stopped. All inbound traffic is thrown away. The only exceptions are PME messages and configuration transactions. If the device is in RC mode, a PM_Turn_Off message can be sent through the PCI Express power management control register.
D2	L1	All outbound traffic is stopped. All inbound traffic is thrown away. The only exceptions are PME messages and configuration transactions. If the device is in RC mode, a PM_Turn_Off message can be sent through the PCI Express power management control register.
D3hot	L1, L2/L3 ready	All outbound traffic is stopped. All inbound traffic is thrown away. The only exceptions are PME messages and configuration transactions. If the device is in RC mode, a PM_Turn_Off message can be sent through the PCI Express power management control register. Note that if a transition of D3 → D0 occurs, a reset is performed to the controller configuration space. In addition, link training restarts.
D3cold	Not supported	Not supported.

### 14.6.4.1 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 or a detection of a beacon or a WAKE# signal from the EP device. The PCI Express controller as an EP device does not support the generation of beacon, so the device can alternatively use one of the GPIO signals as an enable to an external tristate buffer to generate the WAKE# signal if the device needs to wake up from an L2/L3 Ready state. As an RC device, the WAKE# signal from the EP device can be connected to one of the external interrupt input pins to service the WAKE# request if needed.

Figure 14-141 shows an example of how to generate WAKE#.

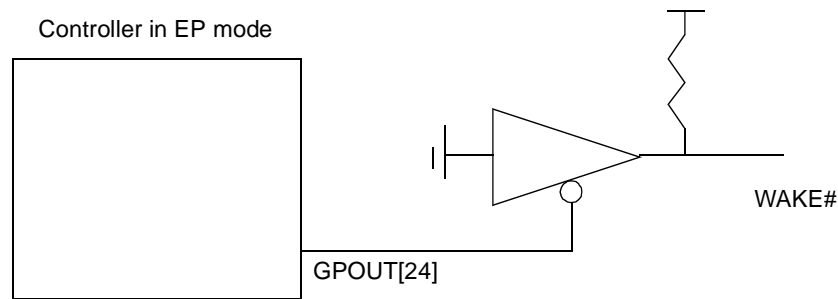


Figure 14-141. Example—How to Generate WAKE#

## 14.6.5 Hot Reset

When a hot reset condition occurs, the controller (in both RC and EP mode) initiates a cleanup of all outstanding transactions and goes into suspend mode. The RC controller then needs to be reset (issuing CBRST in PECR1/PECR2) to bring it back to the idle state followed by a reprogramming of all the CSB bridge registers (offset 0x800–0xFFC, such as the ATMU windows). The EP controller also needs to be reset, followed by a reprogramming of the entire configuration space and CSB bridge registers (offset 0x800–0xFFC, such as the ATMU windows). 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. In EP mode, the device is not permitted to generate a hot reset condition; it can only detect a hot reset condition and initiate the cleanup procedure appropriately.

## 14.7 Initialization/Application Information

The following sections describe initialization sequences for root complex (RC) and end point (EP) modes.

### 14.7.1 Initialization Sequence

The following sequence must be followed. Note the specific stages for RC or EP modes. Upon chip reset, the default SerDes reference clock frequency is assumed to be 100 MHz. Steps 3 and 4 are required only if the reference clock frequency need to be changed to 125 MHz.

1. The device performs its power-on reset sequence. The PCI Express controller and the SerDes PHY are held in reset (controlled by memory mapped registers).
2. Program the system configuration registers of the device, including some PCI Express controller related options (such as local memory windows and clock ratio). See the system configuration registers in [Section 5.1.3.1, “Local Access Register Memory Map,”](#) and the clock configuration registers in [Section 4.5.2, “Clock Configuration Registers.”](#)
3. Set the protocol to PCI Express, the number of lanes, and the reference clock in the SRDSCR4 register. See [Section 15.3.5, “SerDes Control Register 4 \(SRDSCR4\).”](#) In addition, set any SerDes electrical and functional parameters appropriate for PCI Express operation as described in [Chapter 15, “SerDes PHY.”](#)



4. Start the SerDes reset sequence by setting RST field (bit 0) in the SerDes reset control register (SRDSRSTCTL). See [Section 15.3.6, “SerDes Reset Control Register \(SRDSRSTCTL\).”](#)
5. Poll RDONE field (bit 1) in the SerDes reset control register (SRDSRSTCTL).
6. After RDONE is set, wait at least 1 ms.
7. Program the PCI Express control registers 1 and 2. See [Section 5.2.2.11, “PCI Express Control Registers \(PECR1\).”](#) Set the DEV\_TYPE field, to select between EP or RC, take the PCI Express controller out of reset by setting bits [0–2] and optionally select the parameters in the PRI\_DATA, PRI\_DES, and PRI\_PIO fields.
8. Configure the PCI Express core and CSB bridge control registers and address mapping windows to the desired values (that is, inbound/outbound windows).
9. Poll the Status Code field from the LTSSM State Status Register (see [Section 14.4.6.1, “PCI Express LTSSM State Status Register \(PEX\\_LTSSM\\_STAT\)”](#)) to determine when link negotiation is done and link is up (that is, Status Code = 16 link is up).
10. For EP mode only: After system configuration is done, set the CFG\_READY bit in the configuration ready register. See [Section 14.4.6.12, “PCI Express Configuration Ready Register.”](#)
11. For RC mode only: Set the bus master and memory space fields in the PCI Express command register (in the PCI Express configuration space) to allow inbound and outbound transactions. See [Section 14.4.1.3, “PCI Express Command Register.”](#)

#### NOTE

Only in RC mode the local host should perform this programming. When the device is in EP mode, it is expected that the remote PCI Express RC will do this operation by a configuration access.

12. The device is ready to generate or accept PCI Express transactions according to its mode.

## 14.8 DMA Functional Operation

Software uses the DMA engine to initiate memory transfers from the CSB subsystem to the PCI Express system and vice versa without using programmed input/output (PIO), where data is transferred by sending control data through the CPU. The DMA engine provides control registers to enable the transfers. The PCI Express CSB bridge supports two separate engines for read and write DMA operations, referred as RDMA and WDMA, respectively. The DMA engines use a descriptor-based programming model.

#### NOTE

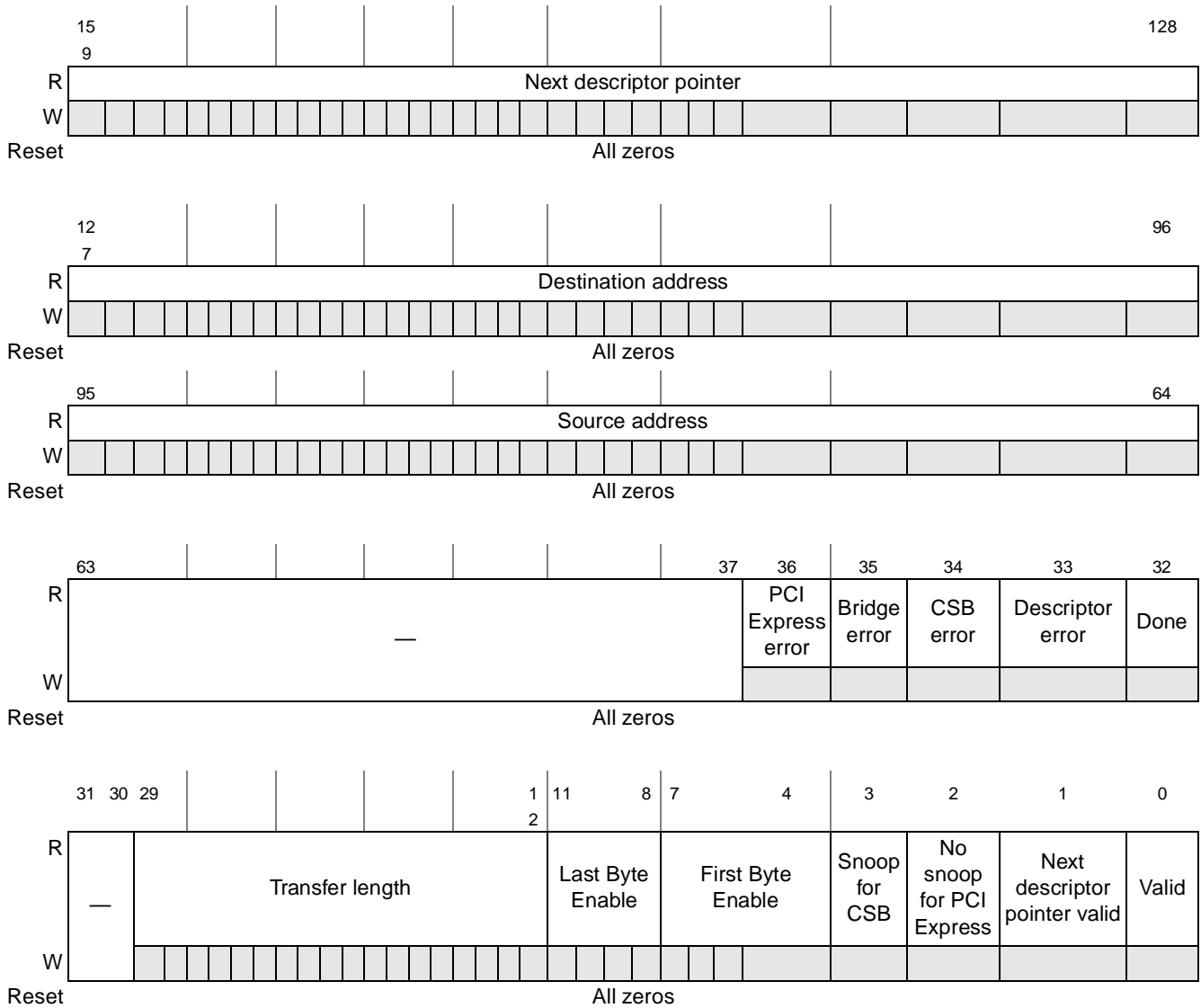
In general, the DMA descriptors use little-endian byte ordering. Software running on the local processor in big-endian mode must byte-swap the data. No byte swapping occurs when the registers are accessed from the PCI Express link.

### 14.8.1 DMA Descriptor Format

The DMA descriptor, shown in [Figure 14-142](#), is five DW wide and consists of control and status fields. The software is responsible to prepare the descriptors at the local memory and program the relevant DMA

control registers. The hardware updates the status register on completion of a DMA data transfer for a descriptor and a chain of descriptors, and report errors.

Offset *Anywhere in local memory* :



**Figure 14-142. DMA Descriptor Format**

Table 14-143 defines the bit fields of a DMA descriptor.

**Table 14-143. DMA Descriptor Bit Fields Description**

Bits	Width	Attribute	Description
159–128	32	Control	Next descriptor pointer. Indicates the location of the next descriptor. Valid only if next_dp is set.
127–96	32	Control	Destination address. Software programs this field to indicate the destination address. For a write DMA, the destination address is a CSB address that is mapped to a PCI Express memory address using the outbound window address translation. For a read DMA, the destination address is the CSB address.

Table 14-143. DMA Descriptor Bit Fields Description (continued)

Bits	Width	Attribute	Description
95–64	32	Control	Source address. Software programs this field to indicate the source address. For a write DMA, the source address is the CSB address. For a read DMA, the source address is a CSB address that is mapped to a PCI Express memory address using the outbound window address translation.
63–37	27	—	Reserved
36	1	Status	PCI Express error. Hardware sets this bit to indicate that a DMA data transfer corresponding to descriptor cannot complete due to a PCI Express error.
35	1	Status	Bridge error. Hardware sets this bit to indicate that DMA data transfer corresponding to descriptor cannot complete due to Bridge error.
34	1	Status	CSB error. Hardware sets this bit to indicate that complete data cannot be fetched due to an CSB Error.
33	1	Status	Descriptor error. Hardware sets this bit to indicate that the next descriptor cannot be fetched due to an CSB error.
32	1	Status	Done. Hardware sets this bit after completing the transaction.
31–30	2	—	Reserved
29–12	18	Control	Transfer length. Software programs this field to indicate the length of transfers in DW or data payload size. A value of zero means that no data is transferred.
11–8	4	Control	Last byte enable indicates the byte enables of the last DW to be transmitted by DMA.
7–4	4	Control	First byte enable. Indicates the byte enables of first DW to be transmitted by DMA.
3	1	Control	Snoop for CSB transactions. 0 The memory transaction is broadcast on the CSB as non-global (that is, not snooped). 1 The memory transaction is broadcast on the CSB as global (that is, snooped)
2	1	Control	No snoop for PCI Express transactions. Indicates the no snoop value to be used in TLP header for PCI Express transactions.
1	1	Control	Next descriptor pointer—valid. Software sets this bit to indicate that a descriptor has a valid next descriptor pointer next_dp. When this bit is cleared, the address of the next descriptor is implicitly specified. This bit must be cleared for block descriptor-based memory.
0	1	Control	Valid. Software sets this bit to indicate that a descriptor is valid and has the information related to data transfer.

## 14.8.2 Write DMA

The PCI Express or CSB software can program the write DMA engines to send data from the CSB system to the PCI Express. After the control registers are programmed, the write DMA engine issues a CSB read request through the DMA read master. To improve system performance, the DMA request is segmented according to a natural aligned CSB address boundary of maximum transfer size (32 bytes).

The entire address space accessed by the DMA controller is prefetchable, so the CSB read request for DMA operation always reads all the bytes. All segments use the same ID and are posted without waiting for the previous read response.

When a response to a CSB read request is received, the segments are packed into the PCI Express memory write request according to the PCI Express MPS. All data requested by the DMA controller is processed as a single data stream as described in this section. For example, when a 256-byte request starting from address 0 is segmented to eight 32-byte CSB read requests, then it is segmented to two 128-byte PCI Express write requests (assuming a 128-byte MPS).

If all data can be packed into one PCI Express write request, no segmentation is performed. Otherwise, the first segment starts from the start address and ends at the MPS address boundary. Byte enables are set according to DMA control register settings. All other segments except the last one start from the MPS address boundary and end at the next boundary. That is, the size of a write request is equal to MPS. All bytes are enabled. Remaining data, if applicable from the last MPS address boundary to the PCI Express end address, is packed into the last segment. Byte enables are set according to DMA control register settings. When all segments get an CSB response with a status of OKAY, the DMA data transfer has completed successfully.

If any data within an CSB read response is aborted by the CSB slave (SLVERR response), all the data received before that point is packed and sent. Any remaining data returned from the CSB slave is discarded, and an error is logged.

If any segment gets a response of DECERR, all data received before that point is packed and sent. Any remaining data returned from the CSB slave is discarded, and an error is logged.

The CSB read master can issue multiple pending CSB read requests if the requests belong to a single DMA request. When all segments are successfully received (that is, they get an CSB response with a status of OKAY), the CSB read master starts processing the next DMA request.

### 14.8.3 Read DMA

The read DMA engines can be programmed by the PCI Express or CSB software to send data from the PCI Express system to the CSB system. After the control registers are programmed, the read DMA engine issues a PCI Express read request. The DMA request is segmented according to the PCI Express MRRS natural aligned address boundary. If all data can be requested in one read request, no segmentation is performed. Otherwise, the first segment starts from the start address and ends at the first MRRS boundary. All other segments except the last one start from the MRRS address boundary and end at the next boundary. That is, the length of the read request is equal to MRRS. Any remaining data, if applicable from the last MRRS address boundary to the end address, is requested in the last segment.

The entire address space accessed by the DMA controller is considered as prefetchable, so a PCI Express read request for DMA operation always enables all byte enables. All segments have a unique tag, so multiple read requests can be pending at any given time.

To improve system performance, when a PCI Express completion comes back, it is segmented according to the CSB maximum size natural aligned address boundary. If all data can be packed into one CSB write request, no segmentation is performed, and the write is performed according to the DMA control register settings. Otherwise, the first segment starts from the start address and ends at the first CSB address boundary. All other segments except the last one start from the CSB address boundary and end at the next boundary. That is, the size of the write request is equal to the CSB maximum size packet. All bytes within each data phase are enabled.

Any remaining data, if applicable from the last CSB address boundary to the PCI Express end address, is packed into the last segment.

Each completion is segmented as a single data stream according to these rules. For example, when a 256-byte request starting from address 0 is converted to two 128-byte PCI Express read requests (assuming a 128-byte MPS/MRRS), the bridge issues two read requests if a tag and a completion buffer are available. When any completion comes back, it is segmented as described in this section. No scatter gather is performed. Because PCI Express can return completions in any order, the bridge may not issue an CSB write request in address order.

When all segments get a response with a PCI Express completion that has a status of successful, the DMA data transfer has completed successfully. If any segment gets a response with a status other than successful, or a completion timeout occurs, the DMA data transfer completes with an error status after all requests complete either normally or abnormally. The data returned for prior requests is still processed and sent to the CSB accordingly.

### 14.8.4 Descriptor-Based DMA

Descriptor-based DMA operation has a specific format in which the host can store information about a data transfer, such as the source address for the data to be transferred, the destination address, the data transfer size, location of the next descriptor, and so on. The host can program a series of descriptors and store them in host local memory. The host also programs the DMA control register to indicate the location of the first descriptor. The DMA control registers are part of the bridge device-specific registers. After programming the control register and descriptors, the host is free to continue with its other functions. The DMA engine is responsible for fetching the descriptor from host memory and moving data from/to host memory.

Software can organize the descriptors in two different formats that are only for reference and do not affect the hardware functionality and requirements:

- Chain descriptor
- Block descriptor

#### 14.8.4.1 Chain Descriptor

Chain descriptors form an  $n$ -way chain in which each descriptor implicitly or explicitly contains the address of the next descriptor. This enables the host to use memory efficiently to store the descriptors even when contiguous memory locations are not available. When the host needs to initiate another transfer, it adds another descriptor in its memory and modifies the pointer of the last descriptor in the chain to the location of the new descriptor. [Figure 14-143](#) illustrates the chain descriptor organization in host memory.

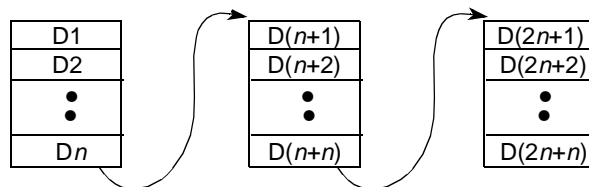


Figure 14-143.  $n$ -Way Chain Descriptor Organization in Host Memory

The number of ways,  $n$ , is software configurable. In this mode,  $n$  descriptors are written in contiguous memory locations. The implicit address of the next descriptor is the next memory location. The last descriptor in the contiguous block contains the explicit address pointer of the next set of  $n$  descriptors. Address 0x0 will never be part of the chain since it should close the chain. Non-contiguous valid descriptors are not supported. If the valid bit of a descriptor in the chain is not set, all of the succeeding descriptors should also have the valid bit as zero.

The software need to follow the following sequence on receiving a chain transfer done interrupt:

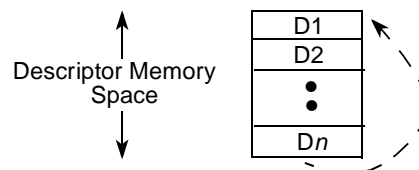
1. Software receives an interrupt for chain transfer done.
2. Software polls the main memory and waits for descriptor done bit to be set (bit 32 in the descriptor). This guarantees that the final data has been written into memory for read DMA. For write DMA, it guarantees that the final data has been sent to PCI Express controller. If a read is sent to the same location, PCI Express controller guarantees that the outbound read will not bypass the outbound write.

The host can reuse the descriptor memory after the DMA/bridge logic processes it. The exact handshake between the hardware and software is described later in this chapter. The DMA registers are described in [Section 14.5.5, “DMA Registers.”](#)

When  $n > 1$ , the hardware uses this knowledge to prefetch descriptors in advance, thereby reducing possible holes in transmission. This might be useful for applications that request several small DMA transfer requests, such as an Ethernet traffic. For applications that request large data transfers, the effect on performance due to descriptor fetching is not significant.

#### 14.8.4.2 Block Descriptors

Block descriptors are a special case of chain descriptor in which all the descriptors are part of a single array. A portion of host memory is reserved to store the descriptors. The starting address of this block is indicated in the DMA control register. This block of memory serves as a circular buffer. Software writes the first descriptor to the first address in the descriptor block address space. Subsequent descriptors are written in the consecutive locations until the last location in the descriptor space. After that, the next descriptor is again written into the first location. All descriptors except the one written into the last descriptor location in the block have an implicit address that points to the immediate next descriptor location in the block. The last descriptor contains an explicit address pointer to the first descriptor location of the block. Software must ensure that a descriptor is processed by hardware before overwriting it. [Figure 14-144](#) illustrates the organization of the block descriptors in memory.



**Figure 14-144. Block Descriptor Organization in Host Memory**

The DMA can fetch prefetch a configurable number of descriptors in a single shot. This number depends on the chain organization ( $n$ ) as well as the number of descriptor registers in the DMA engine.

### 14.8.4.3 Descriptor Format

For more information on descriptor format, refer to [Section 14.8.1, “DMA Descriptor Format.”](#) The fetched descriptors are stored in the bridge configuration space registers. Each time the DMA controller fetches a new set of descriptors, the register is updated to indicate the value/fields of the descriptors. After completion of a transaction specified by a descriptor, the status fields are updated in the descriptor registers in the configuration space. Also, the descriptor status is written back into host local memory and an interrupt is generated if enabled. When the last of  $n$  descriptors is processed, the next set of descriptors is fetched from memory.

### 14.8.4.4 Software-Hardware Handshake

Hardware and software communicate through descriptors, control registers, and interrupts. The control register programmed by software indicates to hardware the location of the first descriptor. The descriptors programmed by software in host memory or directly in the descriptor register provide information to the hardware about the impending data transfer. After the initial descriptor location and other DMA parameters are programmed in the DMA control registers, software sets the ‘start’ bit in the control register to trigger the DMA controller to initiate the operation. When it detects that the ‘start’ bit is set, the DMA controller uses these programmed DMA parameters to initiate a descriptor fetch and the corresponding data transfer, and then it clears the “start” bit.

The DMA controller executes the transfer according to the instructions given in the descriptor and then updates both the descriptor and DMA status register to indicate the status of the transfer operation. Descriptor status is updated in host local memory. An interrupt is also generated to the host, if enabled.

At any time, software can parse the done bit in the descriptor chain located in host local memory to determine the point to which the DMA controller has executed and also if the descriptors were successfully completed. The status register also gives details about transfer status, including details about errors if any occurred.

If the DMA encounters an unprogrammed descriptor (ready = 0) in the  $n$  descriptor array that it fetched, it first executes any remaining prefetched valid descriptors and then sends an interrupt to the host, if enabled. The transfers can resume in one of two ways:

- The DMA controller automatically fetches the same set of descriptors again after the time indicated in PEX\_DMA\_DSTMR[DSRT] (see [Section 14.5.2.2, “PCI Express DMA Descriptor Timer Register \(PEX\\_DMA\\_DSTMR\)”](#)). If the descriptor is ready now, it continues execution or else checks again later.
- Software can force immediate resuming of the transfers by disabling and then re-enabling the DMA.

When the DMA controller completes the data transfer for the last descriptor in the chain (null descriptor), it updates the descriptor and status register. An interrupt is generated, if enabled. The DMA engine moves into the IDLE state until software re-triggers it by setting the ‘start’ bit in the control register.





# Chapter 15

## SerDes PHY

### 15.1 Introduction

The SerDes PHY block includes the following components:

- SerDes PHY
- Protocol multiplexer and converter per protocol
- Control registers and control logic
- Power-down/reset state machine for cold (power-on) or warm (software-initiated) reset of the SerDes, PCVTR, and controllers
- Interface with the clock controls

#### 15.1.1 Overview

Figure 15-1 is a block diagram showing the functional blocks inside the SerDes PHY block and its connections to other modules in the processor.

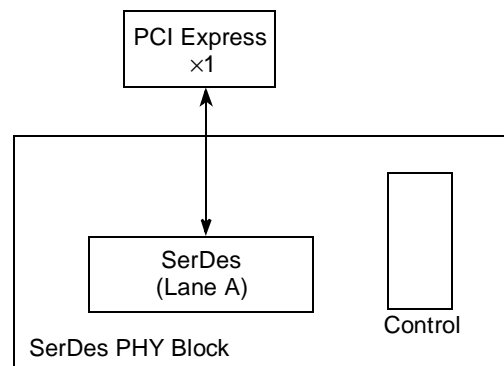


Figure 15-1. SerDes PHY Block Diagram

#### 15.1.2 Features

The SerDes PHY block has the following features:

- Support for one ×1 PCI Express controller
- Link-layer interfaces to IP controller
- Memory-mapped registers with 256-byte address region
- SerDes power-down/reset state machine for cold (power-on) or warm (software-initiated) reset of SerDes, PHY, and controllers

### 15.1.3 Mode of Operation

The SerDes PHY block supports one lane (Lane A) running  $\times 1$  PCI Express at 2.5 Gbps as the mode of operation.

#### NOTE

SerDes block's second lane, Lane B, is externally not available and therefore, should be powered down. For more information, see [Section 15.3.2, "SerDes Control Register 1 \(SRDSCR1\)."](#)

### 15.1.4 Clock

The SerDes control has one clock which is running at platform speed. This is an internally generated clock based off of the system clock.

## 15.2 External Signals

[Table 15-1](#) describes the external signals to the SerDes PHY block.

**Table 15-1. SerDes External Signals—Detailed Signal Descriptions**

Signal	I/O	Description
SD_IMP_CAL_RX	I	Receiver impedance calibration control signal. This pin requires an external resistor to ground to set the differential input impedance of the receivers.
		<b>State Meaning</b> Assertion/Negation—The SerDes acts as an integrated active impedance calibration circuit to ensure the best possible impedance control of the receiver's link termination resistors. The calibration circuit uses an externally established impedance against which internal impedance is calibrated. The user connects a 200 $\Omega$ , 1% tolerance resistor between the sd_imp_cal_rx input and ground. If the user wishes to set the impedance control to its nominal value, the sd_imp_cal_rx input maybe tied directly to the xcorevdd supply.
SD_IMP_CAL_TX	I	Transmitter impedance calibration control signal. This pin requires an external resistor to ground to set the differential output impedance of the transmitters.
		<b>State Meaning</b> Asserted/Negated—The SerDes acts as an integrated active impedance calibration circuit to ensure the best possible impedance control of the transmitter's output impedance resistors. The calibration circuit uses and externally established impedance against which internal impedance is calibrated. The user connects a 100 $\Omega$ , 1% tolerance resistor between the sd_imp_cal_tx input and ground. If the user wishes to set the impedance control to its nominal value, the sd_imp_cal_tx input maybe tied directly to the xpadvdd supply.
SD_REF_CLK	I	SerDes PLL reference clock, along with $\overline{\text{SD\_REF\_CLK}}$ , is used by the SerDes PLL to generate all of the necessary clocks for the SerDes.
		<b>Timing</b> Assertion/Negation—Choices of input reference clock values are limited by specification, output bit rate and PLL functionality.
$\overline{\text{SD\_REF\_CLK}}$	I	SerDes PLL reference clock complement, along with SD_REF_CLK, is used by the SerDes PLL to generate all of the necessary clocks for the SerDes.
		<b>Timing</b> Assertion/Negation—Choices of input reference clock values are limited by specification, output bit rate and PLL functionality.

**Table 15-1. SerDes External Signals—Detailed Signal Descriptions (continued)**

Signal	I/O	Description
RXA	I	Serial receiver data, lane A, positive.
		<b>Timing</b> Assertion/Negation—Serial differential receiver input which can be configured to meet the PCI Express specifications.
$\overline{\text{RXA}}$	I	Serial receiver data, lane A, complement.
		<b>Timing</b> Assertion/Negation—Serial differential receiver input which can be configured to meet the PCI Express specifications.
TXA	O	Serial transmitter data, lane A, positive.
		<b>Timing</b> Assertion/Negation—Serial differential transmitter output which can be configured to meet the PCI Express specifications.
$\overline{\text{TXA}}$	O	Serial transmitter data, lane A, complement.
		<b>Timing</b> Assertion/Negation—Serial differential transmitter output which can be configured to meet PCI Express specifications.

## 15.3 Memory Map/Registers

Table 15-2 lists the SerDes PHY block registers and their addresses. Reading undefined portions of the memory map returns all zeros; writing has no effect. All the registers are 32 bits wide.

**Table 15-2. SerDes PHY Block Memory Map**

Offset	Register	Access	Reset	Section/Page
<b>SerDes PHY—Block Base Address 0xE_3000</b>				
0x000	SRDSCR0—SerDes Control Register 0	R/W	0x1100_CC30	<a href="#">15.3.1/15-4</a>
0x004	SRDSCR1—SerDes Control Register 1	R/W	0x0000_0040	<a href="#">15.3.2/15-6</a>
0x008	SRDSCR2—SerDes Control Register 2	R/W	0x0080_0000	<a href="#">15.3.3/15-7</a>
0x00C	SRDSCR3—SerDes Control Register 3	R/W	0x0101_0000	<a href="#">15.3.4/15-8</a>
0x010	SRDSCR4—SerDes Control Register 4	R/W	0xnn00_0n0n	<a href="#">15.3.5/15-9</a>
0x014–0x01C	Reserved	—	—	—
0x020	SRDSRSTCTL—SerDes Reset Control Register	R/W	0x0044_4500	<a href="#">15.3.6/15-10</a>
0x024–0x1FC	Reserved	—	—	—

### NOTE

Reserved bits should always be written with the value they return when read. That is, the register should be programmed by reading the value, modifying appropriate fields, and writing back the value.

### 15.3.1 SerDes Control Register 0 (SRDSCR0)

SRDSCR0, shown in Figure 15-2, contains the functional control bits for the SerDes logic.

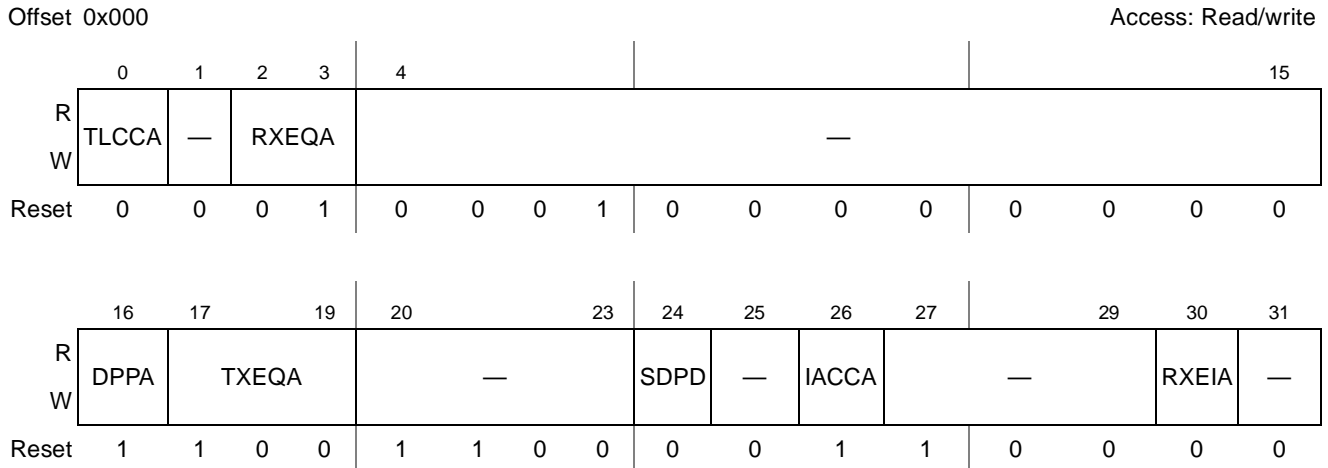


Figure 15-2. SerDes Control Register 0 (SRDSCR0)

Table 15-3 defines the bit fields of SRDSCR0.

Table 15-3. SRDSCR0 Field Descriptions

Bits	Name	Description
0	TLCCA	Tracking loop centering control for lane A. When enabled, it centers the first-stage digital filter after the second-stage filter moves transition point. 0 Enable recentering algorithm 1 Disable recentering algorithm Recommended setting per protocol is PCI Express: 0
1	—	Reserved <sup>1</sup>
2–3	RXEQA	Receive equalization selection bus for lane A—when asserted in PCI Express mode: 00 No equalization 01 2 dB of equalization 10 4 dB of equalization 11 Reserved
4–15	—	Reserved <sup>1</sup>
16	DPPA	Diff pk-pk swing for lane A. Sets the peak value for output swing of transmitters and the amount of transmit equalization for lane A. 0 V <sub>DD</sub> -diff-pk-pk 1 Reserved Recommended setting per protocol is PCI Express: 0

Table 15-3. SRDSCR0 Field Descriptions (continued)

Bits	Name	Description
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. 000 No equalization 001 1.09× relative amplitude 010 1.2× relative amplitude 011 1.33× relative amplitude 100 1.5× relative amplitude 101 1.71× relative amplitude 110 2.0× relative amplitude 111 Reserved Recommended setting per protocol is PCI Express: 100
20–23	—	Reserved <sup>1</sup>
24	SDPD	SerDes 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. 0 Application mode 1 Block power down
25	—	Reserved <sup>1</sup>
26	IACCA	Used to set on-chip AC coupling in the receiver in lane A. 0 Disable on-chip AC coupling 1 Enable on-chip AC coupling Recommended setting per protocol is PCI Express: 1
27–29	—	Reserved <sup>1</sup>
30	RXEIA	When asserted, places lane A into receiver electrical idle state. 0 Lane A is not 'forced' into receive electrical idle state 1 Place lane A into electrical idle state Recommended setting per protocol is PCI Express: 0
31	—	Reserved <sup>1</sup>

<sup>1</sup> Bits with reset value of one must be written as one during any write operation.

### NOTE

While writing to this register, bit 7, 20, 21, and 27 should be set to 1.

## 15.3.2 SerDes Control Register 1 (SRDSCR1)

SRDSCR1, shown in Figure 15-3, contains the functional control bits for the SerDes logic.

Lane A can be powered down using SRDSCR1[0]. The entire SerDes must be reset to activate a lane from power-down. Refer to Section 15.3.6, “SerDesn Reset Control Register (SRDSRSTCTL).”

Offset 0x04

Access: Read/write

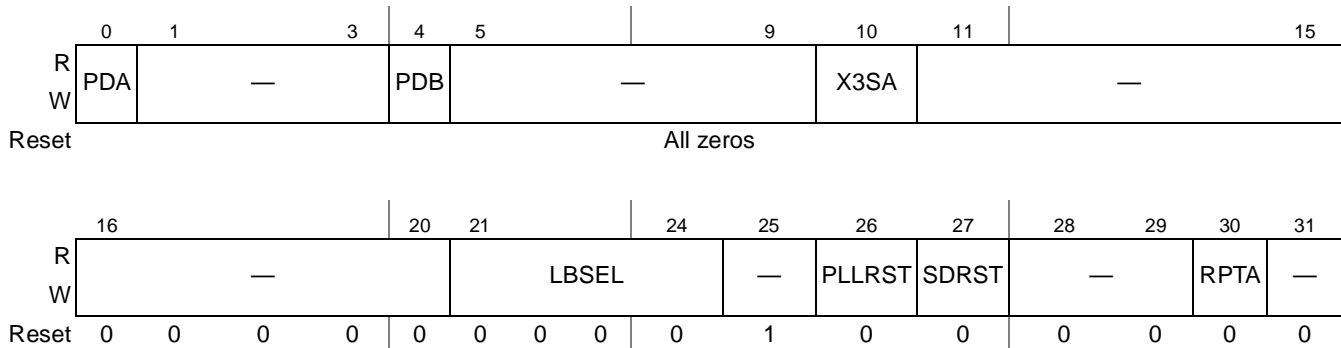


Figure 15-3. SerDesn Control Register 1 (SRDSnCR1)

Table 15-4 describes the SRDSCR1.

Table 15-4. SRDSCR1 Field Descriptions

Bits	Name	Description
0	PDA	Lane A power down. 0 Normal 1 Power down lane A
1–3	—	Reserved
4	PDB	Lane B power down 0 Upon reset 1 Power down lane B
5–9	—	Reserved
10	X3SA	Lane A transmitter three-state. 0 Normal 1 The transmitter output is disabled and place in a three-state condition
11–20	—	Reserved
21–24	LBSEL	Select type loop-back. 0000 Application mode 0001 Digital loopback 0010 Analog loopback All other modes reserved for test
25	—	Reserved
26	PLLRST	PLL master reset for SerDes. Resets the PLL and the impedance calibration circuitry. software needs to set and clear this bit. 0 Application mode 1 Reset <b>Note:</b> PLLRST can also be done by setting SRDSnRSTCTL[RST]. In this case, PLLRST can self-clear.

Table 15-4. SRDSCR1 Field Descriptions (continued)

Bits	Name	Description
27	SDRST	Master reset for SerDes logic. Resets all logic in SerDes lane A. Software needs to set and clear this bit. 0 Application mode 1 Reset <b>Note:</b> SDRST can also be done by setting SRDS $\overline{n}$ RSTCTL[RST]. In this case, SDRST can self-clear.
28–29	—	Reserved
30	RPTA	To enable repeater mode on lane A. Enables data received on serial inputs to be repeated back through to the transmitter outputs after data sampling and transition recovery on lane A. 0 Repeater mode disabled 1 Enable repeater mode on lane A
31	—	Reserved

### 15.3.3 SerDes Control Register 2 (SRDSCR2)

SRDSCR2, shown in Figure 15-4, contains the functional control bits used for the SerDes logic.

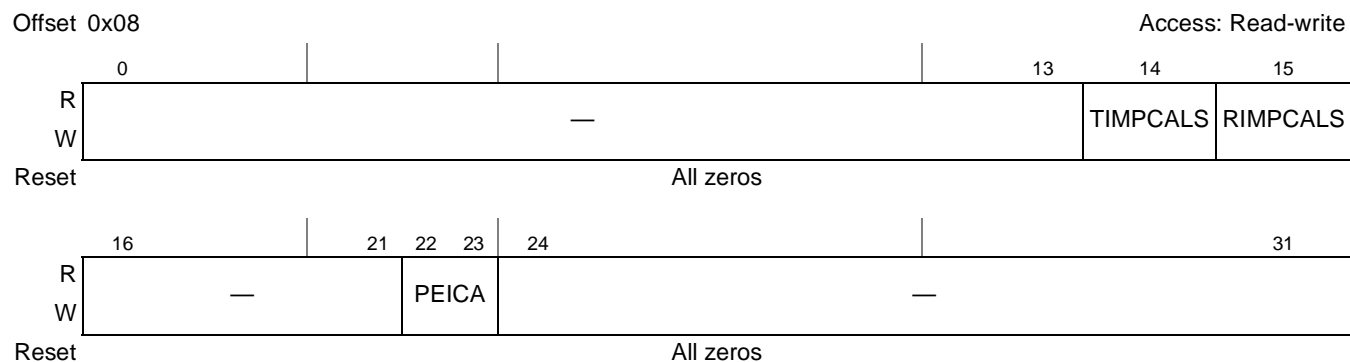


Figure 15-4. SerDes Control Register 2 (SRDSCR2)

Table 15-5 describes the SRDSCR2.

Table 15-5. SRDSCR2 Field Descriptions

Bits	Name	Description
0–13	—	Reserved
14	TIMPCALS	Transmitter impedance calibration stop command. Allows user to stop calibration of transmitter impedances. 0 Run transmit impedance calibration 1 Stop transmit impedance calibration Recommended setting per protocol is PCI Express: 0
15	RIMPCALS	Receiver impedance calibration stop command. Allows user to stop calibration of receiver impedances. 0 Run receive impedance calibration 1 Stop receive impedance calibration Recommended setting per protocol is PCI Express: 0
16–21	—	Reserved

Table 15-5. SRDSCR2 Field Descriptions (continued)

Bits	Name	Description
22–23	PEICA	PCII-EXP Receiver electrical idle detection control 00 Exit from idle ~85 UI and unexpected idle detect ~1 $\mu$ s (application mode) 01 Exit from idle ~85 UI and unexpected idle detect ~10 $\mu$ s 10 Exit from idle ~45 UI and unexpected idle detect ~1 $\mu$ s 11 Bypass
24–31	—	Reserved

### 15.3.4 SerDes Control Register 3 (SRDSCR3)

SRDSCR3, shown in Figure 15-5, contains the functional control bits for the SerDes logic.

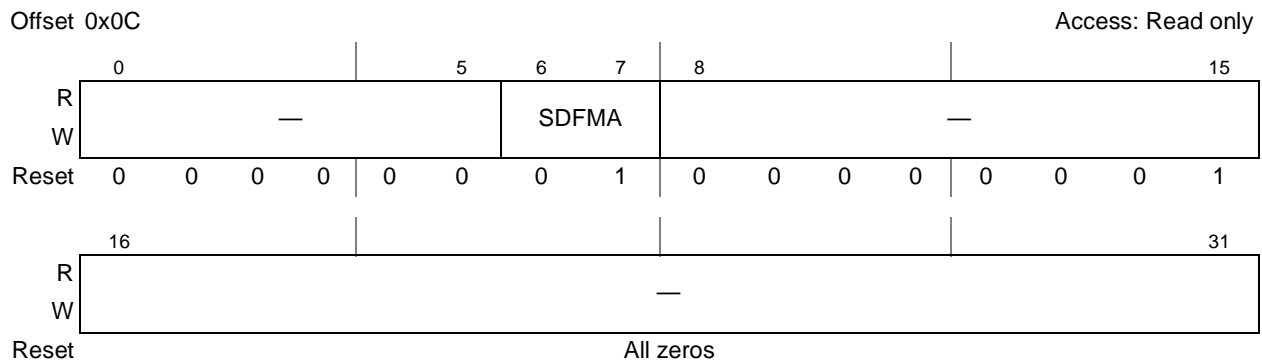


Figure 15-5. SerDes Control Register 3 (SRDSCR3)

Table 15-6 describes the SRDSCR3.

Table 15-6. SRDSCR3 Field Descriptions

Bits	Name	Description
0–5	—	Reserved <sup>1</sup>
6–7	SDFMA	Sets the bandwidth of the digital filter to optimize for given frequency offset specification for lane A. 00 Reserved 01 600 ppm (PCI Express) 10 Reserved 11 Reserved Recommended setting per protocol is PCI Express: 01
8–31	—	Reserved <sup>1</sup>

<sup>1</sup> Bits with reset value of one must be written as one during any write operation.

#### NOTE

While writing to this register, bit 15 should be set to 1.



### 15.3.5 SerDes Control Register 4 (SRDSCR4)

SRDSCR4, shown in Figure 15-6, contains the functional control bits for the SerDes logic.

#### NOTE

To power down lane A, use SRDSCR1[0] (refer to Section 15.3.2, “SerDes Control Register 1 (SRDSCR1)”).

The valid combination for protocol select is PCI Express mode (PROTA = 001), which is only an ×1 lane. The reference clock can be either 100 or 125 MHz.

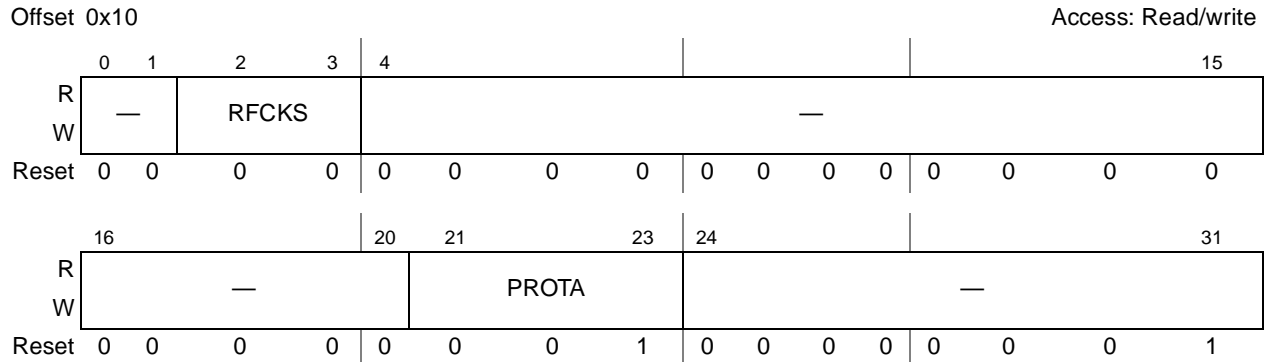


Figure 15-6. SerDes Control Register 4 (SRDSCR4)

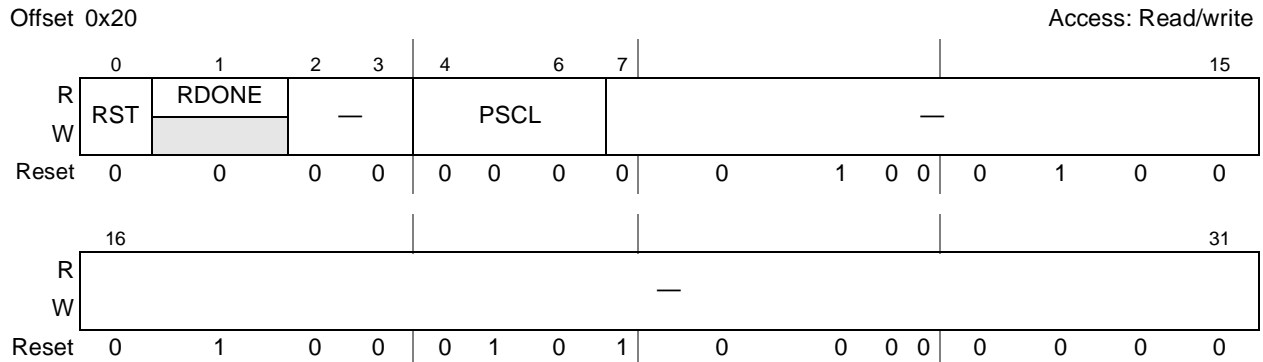
Table 15-7 describes the SRDSCR4 bit fields.

Table 15-7. SRDSCR4 Field Descriptions

Bits	Name	Description
0–1	—	Reserved
2–3	RFCKS	SerDes reference clock selection. 00 100 MHz 01 125 MHz 10 Reserved 11 Reserved
4–20	—	Reserved
21–23	PROTA	Lane A protocol select (PCI Express) 001 PCI Express at 2.5 Gbps All other modes are reserved.
24–31	—	Reserved

### 15.3.6 SerDes $n$ Reset Control Register (SRDSRSTCTL)

SRDSRSTCTL, shown in [Figure 15-7](#), contains the control for SerDes reset state machine counter values.



**Figure 15-7. SerDes Reset Control Register (SRDSRSTCTL)**

[Table 15-8](#) describes the SRDSRSTCTL register.

**Table 15-8. SRDSRSTCTL Field Descriptions**

Bits	Name	Description
0	RST	To initiate SerDes soft reset software writes a 1. SerDes reset state machine clears bit when reset is done. Software can only assert this bit but not clear it. If cleared in the middle, the reset state machine ignores the change.
1	RDONE	SerDes reset done from SerDes state machine. When this bit is set, the SerDes is ready to start link training in concert with the protocol controller.
2–3	—	Reserved
4–6	PSCL	Determines how many platform cycles equal one state machine tick. This value should be changed only when SRDS $n$ RSTCTL[RST] is set. 000 Up to 200 MHz platform (1 platform cycle per tick) 001 Up to 400 MHz platform (2 platform cycle per tick) All other combinations are reserved
7–31	—	Reserved

## 15.4 Initialization Sequence and Reset

SerDes and PHY can be initialized by software anytime by doing a reset\_req. This is done by setting the SRDSRSTCTL[0] register field. Setting this register field starts the warm reset state machine. Software does not need to clear this bit.

Register fields to configure before software reset request:

- SRDSRSTCTL register
  - Configure prescale and counter values (refer to [Figure 15-7](#))
- SRDSCR3 Register
  - SRDSCR3[6–7], SRDSCR3[14–15]—configure to 0x01 in PCI Express mode

- SRDSCR4 register
  - Configure protocol select, reference clock frequency, and PCI Express ×1 lane (refer to [Figure 15-6](#))

Valid combination for protocol select is:

- PCI Express mode (AD\_PROTO\_SEL/EH\_PROTO\_SEL[2–0] = 001) is only an ×1 lane. The reference clock can be either 100 or 125 MHz.

#### NOTE

- The entire SerDes need to be reset in order to activate lane A from power-down. The SRDSCR4 register is initialized base on RCWH[TSEC1M] and RCWH[TSEC2M]. The SRDSCR3 register is not initialized based on RCWH[TSEC1M] and RCWH[TSEC2M]; it needs to be done explicitly based on the usage scenario. (Refer to [Section 4.3.2.2.4, “eTSEC1 Mode.”](#))
- For reset from software, the recommended option is to use SRDSRSTCTL[RST], and let the hardware control the timing of the various SERDES resets and power-downs. Software can poll SRDSRSTCTL[RDONE] to determine when the reset is complete. For more information, see [Section 15.3.6, “SerDesn Reset Control Register \(SRDSRSTCTL\).”](#)

## 15.5 Power Management: Power Down

The SerDes is capable of several different power management states depending on the settings of the protocol selection and power down signals.

By setting the register field SRDSCR0[24] powers down the entire SerDes and is comparable to the PCI Express L2 low power link state. The steps for powering down the SerDes are as follows:

1. Apply power to all XCOREVDD, XPADVDD, SDAVDD supplies.
2. Be sure all XCOREVSS, XPADVSS, and SDAVSS supplies are grounded.
3. Assert the SRDSCR0[24] control from the chip logic to whichever SerDes block is not in use. This safely parks all the analog circuitry and stops all SerDes-generated clocks.
4. Tie all unused RX<sub>n</sub> and  $\overline{\text{RX}}_n$  serial differential inputs to XCOREVSS.
5. Float all unused TX<sub>n</sub> and  $\overline{\text{TX}}_n$  serial differential outputs.
6. If a SerDes block is not being used and has its own receiver and transmitter calibration external resistors, then these can be tied to XCOREVDD for the receiver (SD\_IMP\_CAL\_RX) and XPADVDD for the transmitter (SD\_IMP\_CAL\_TX).
7. Tie SD\_REF\_CLK and  $\overline{\text{SD\_REF\_CLK}}$  both to XCOREVSS.

#### NOTE

- If the entire SerDes is powered down, or even if parts of the SerDes is powered down, all power pads in the SerDes must be powered.

- The unused lane B of SerDes may be powered down by programming PDB bit in SRDSCR1 to 1. For more information, see [Section 15.3.2, “SerDes Control Register 1 \(SRDSCR1\).”](#)

Powering down the SerDes includes the following steps:

1. Disable the PLL and place its output clocks into a known, static state.
2. Power down the receiver termination and amplifier cells. In PCI Express mode, there is still a differential termination, but its value is no longer calibrated. Also, there is no longer a termination to ground.
3. Power down the transmitter and receiver impedance control amplifiers so that the termination impedances are uncalibrated.
4. Power down the transmitter cell. The  $V_{TX-DIFFp} < 20$  mV. The DC common mode voltage is not held.

# Chapter 16

## Enhanced Three-Speed Ethernet Controllers

### 16.1 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. 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. Two 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 II Pro TSEC, allowing existing driver software to be re-used with minimal change. Each eTSEC is organized as shown in [Figure 16-1](#).

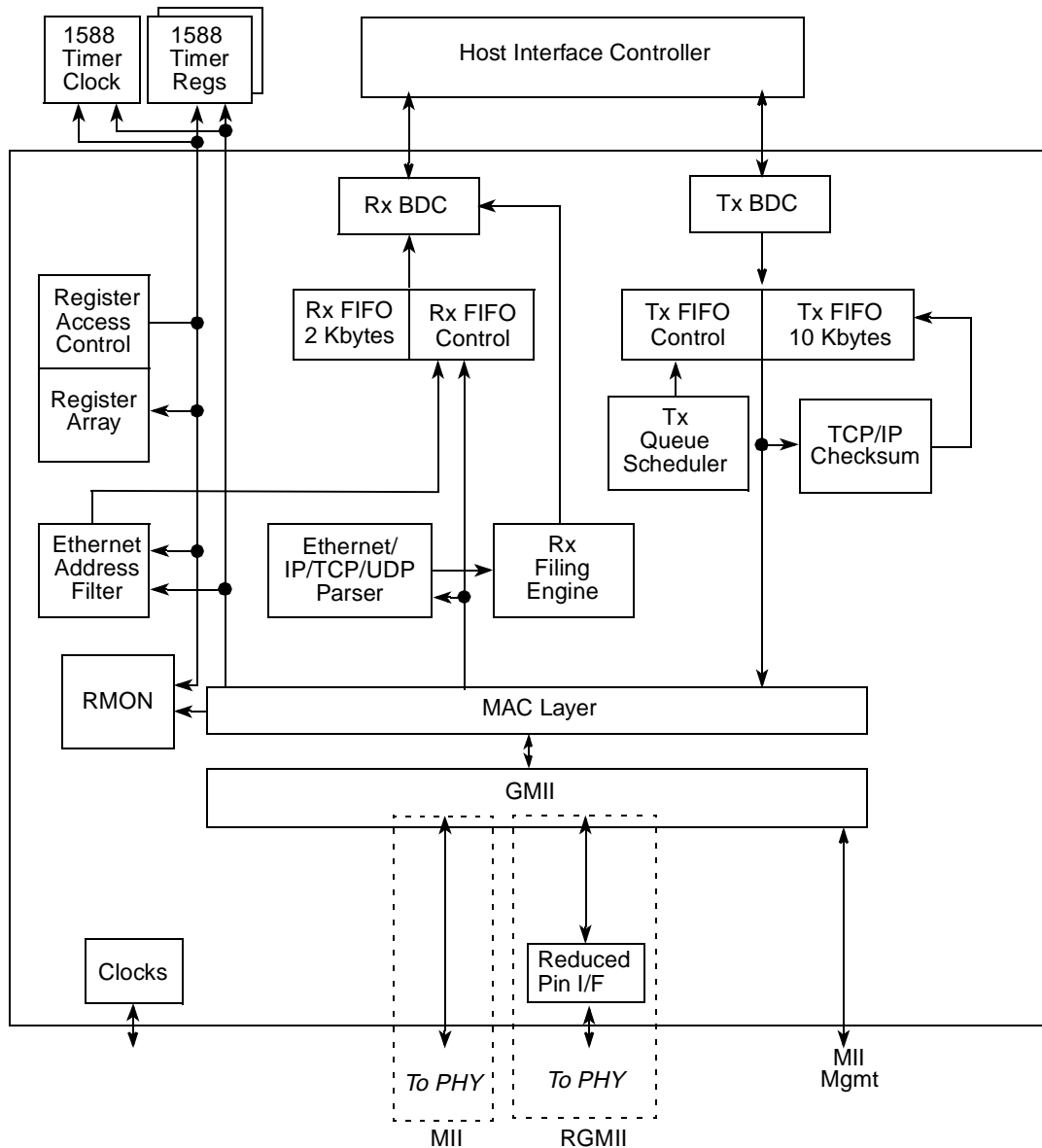


Figure 16-1. eTSEC Block Diagram

## 16.2 Features

The eTSECs of the device include these distinctive features:

- IEEE 802.3, 802.3u, 802.3x, 802.3z, 802.3ac, 802.3ab compatible
- Support for different Ethernet physical interfaces:
  - 10/100 Mbps IEEE 802.3 MII
  - 10/100 Mbps RGMII
  - 1000 Mbps full-duplex RGMII
- 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 MPC8349E (PowerQUICC II Pro) TSEC
  - PowerQUICC II Pro 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 II Pro TSEC subset
  - TSEC\_ID register permits TSEC versus enhanced TSEC differentiation
- Hardware assist for 1588 compliant timestamping
  - 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

## 16.3 Modes of Operation

The eTSEC's primary operational modes are the following:

- 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<sub>n</sub>\_TX\_CLK and TSEC<sub>n</sub>\_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.

- 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 16.6.2.7, “Frame Recognition.”](#)

- 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 16.6.3, “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 16.6.4, “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 16.6.7.2, “Transmit Data Buffer Descriptors \(TxBD\),”](#) enable these options and operate with parameters prepended to frame buffers, as described in [Section 16.6.3, “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 16.5.3.2.1, “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 16.7.1, “Interface Mode Configuration,”](#) for details.

## 16.4 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[3–0] 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 RGMII option is reduced-pin implementations of the GMII.
- 1588 timer signals.

Table 16-1 lists the network interface signals.

**Table 16-1. eTSEC<sub>n</sub> Network Interface Signal Properties**

Signal Name	Function	Reset State
TSEC <sub>n</sub> _COL	MII—collision, input	—
TSEC <sub>n</sub> _CRS	MII—carrier sense, input	—
TSEC <sub>n</sub> _GTX_CLK	RGMII—inverted transmit clock feedback, output MII—transmit clock feedback when transmission is enabled, zero otherwise, output	0
TSEC <sub>n</sub> _GTX_CLK125	Oscillator source for RGMII transmit clock, input, shared by all eTSECs	—
TSEC_MDC	Management clock, output.	0
TSEC_MDIO	Management data, bidirectional.	Hi-Z (input)
TSEC <sub>n</sub> _RX_CLK	MII, RGMII—receive clock, input	—
TSEC <sub>n</sub> _RX_DV	MII—receive data valid, input RGMII (RX_CLK rising)—receive data valid, input RGMII (RX_CLK falling)—receive error, input	—
TSEC <sub>n</sub> _RXD[3:0]	MII—Receive data bits 3:0, input RGMII (RX_CLK rising) —Receive data bits 3:0, input RGMII (RX_CLK falling)—Receive data bits 7:4, input	—
TSEC <sub>n</sub> _RX_ER	MII—Receive error, input RGMII—Unused	—
TSEC <sub>n</sub> _TX_CLK	MII—transmit clock, input RGMII—unused	—
TSEC <sub>n</sub> _TXD[3:0]	MII—Transmit data bits 3:0, output RGMII (TX_CLK rising)—Transmit data bits 3:0, output RGMII (TX_CLK falling)—Transmit data bits 7:4, output	0000
TSEC <sub>n</sub> _TX_ER	MII—transmit error, output RGMII—unused, output driven zero	0
TSEC <sub>n</sub> _TX_EN	MII—Transmit data valid, output RGMII (TX_CLK rising)—Transmit data enabled, output RGMII (TX_CLK falling)—Transmit error, output	0
TSEC_TMR_CLK	1588—Clock input External high precision timer reference clock input (chip external input pin).	—
TSEC_TMR_GCLK	1588—Clock output Phase aligned timer clock divider output (chip external output pin).	0
TSEC_TMR_TRIG1	1588—Trigger in 1 External timer trigger input 1. This is an asynchronous general purpose input (chip external input pin).	—

Table 16-1. eTSEC<sub>n</sub> Network Interface Signal Properties (continued)

Signal Name	Function	Reset State
TSEC_TMR_TRIG2	1588—Trigger in 2 External timer trigger input 2. This is an asynchronous general purpose input (chip external input pin).	—
TSEC_TMR_PP1	1588—Pulse out 1 Timer pulse per period 1. It is phase aligned with 1588 timer clock (chip external output pin).	0
TSEC_TMR_PP2	1588—Pulse out 2 Timer pulse per period 2. It is phase aligned with 1588 timer clock (chip external output pin).	0
TSEC_TMR_PP3	1588—Pulse out 3 Timer pulse per period 3. It is phase aligned with 1588 timer clock (chip external output pin).	0
TSEC_TMR_ALARM1	1588—Alarm out 1	0
TSEC_TMR_ALARM2	1588—Alarm out 2	0
TSEC <sub>n</sub> _TMR_RX_ESFD	1588—Receive external start of frame delimiter	—
TSEC <sub>n</sub> _TMR_TX_ESFD	1588—Transmit external start of frame delimiter	—

### 16.4.1 Detailed Signal Descriptions

Table 16-2 is a description of the eTSEC interface signals. All other modes follow the IEEE 802.3 standard, 2000 edition. Input signals not used are internally disabled. Except for TSEC<sub>n</sub>\_GTX\_CLK, output signals not used are driven low.

#### NOTE

For more information on RGMII mode, see *Hewlett-Packard Reduced Gigabit Media-Independent Interface (RGMII) Specification, Version 1.2a, Dated 9/22/2000*.

Table 16-2. eTSEC Signals—Detailed Signal Descriptions

Signal	I/O	Description
TSEC <sub>n</sub> _COL	I	Collision input. The behavior of this signal is not specified while in full-duplex mode.
		<b>State Meaning</b> Asserted/Negated—In MII mode, this signal is asserted upon detection of a collision, and must remain asserted while the collision persists. This signal is not used in the RGMII mode.
		<b>Timing</b> Asserted/Negated—This signal is not required to transition synchronously with TSEC <sub>n</sub> _TX_CLK or TSEC <sub>n</sub> _RX_CLK.
TSEC <sub>n</sub> _CRS	I	Carrier sense input. This signal is not used in the RGMII mode.
		<b>State Meaning</b> Asserted/Negated—In MII mode, TSEC <sub>n</sub> _CRS is asserted while the transmit or receive medium is not idle. In the event of a collision, TSEC <sub>n</sub> _CRS must remain asserted for the duration of the collision.
		<b>Timing</b> Asserted/Negated—This signal is not required to transition synchronously with TSEC <sub>n</sub> _TX_CLK or TSEC <sub>n</sub> _RX_CLK.

Table 16-2. eTSEC Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description
TSEC <sub>n</sub> _GTX_CLK	O	Gigabit transmit clock. This signal is an output from the eTSEC into the PHY. In RGMII mode, TSEC <sub>n</sub> _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. This signal feeds back the uninverted transmit clock in MII mode, but feeds back an inverted transmit clock in RGMII mode. This signal is driven low unless transmission is enabled.
TSEC_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. TSEC_GTX_CLK125 is a 125-MHz input into the eTSEC and is used to generate all 125-MHz related signals and clocks in the RGMII mode. This input is not used in the MII mode.
TSEC_MDC	O	Management data clock. This signal is a clock (typically 2.5 MHz) supplied by the MAC (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.
TSEC_MDIO	I/O	Management data input/output.
		<b>State Meaning</b> Asserted/Negated—TSEC_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.
		<b>Timing</b> Asserted/Negated—This signal is required to be synchronous with the TSEC_MDC signal.
TSEC <sub>n</sub> _RX_CLK	I	Receive clock. In MII or RGMII mode, the receive clock TSEC <sub>n</sub> _RX_CLK is a continuous clock (2.5, 25, or 125 MHz) that provides a timing reference for TSEC <sub>n</sub> _RX_DV, TSEC <sub>n</sub> _RXD, and TSEC <sub>n</sub> _RX_ER.
TSEC <sub>n</sub> _RX_DV	I	Receive data valid. In MII mode, if TSEC <sub>n</sub> _RX_DV is asserted, the PHY is indicating that valid data is present on the MII interface. In RGMII mode, TSEC <sub>n</sub> _RX_DV becomes RX_CTL. The RX_DV and RX_ERR are received on this signal on the rising and falling edges of TSEC <sub>n</sub> _RX_CLK.
TSEC <sub>n</sub> _RXD[3:0]	I	Receive data in. In MII mode, TSEC <sub>n</sub> _RXD[3:0] represents a nibble of data to be transferred from the PHY to the MAC when TSEC <sub>n</sub> _RX_DV is asserted. A completely-formed SFD must be passed across the MII. While TSEC <sub>n</sub> _RX_DV is not asserted, TSEC <sub>n</sub> _RXD has no meaning. In RGMII mode, data bits 3–0 are received on the rising edge of TSEC <sub>n</sub> _RX_CLK and data bits 7–4 are received on the falling edge of TSEC <sub>n</sub> _RX_CLK.
TSEC <sub>n</sub> _RX_ER	I	Receive error
		<b>State Meaning</b> Asserted/Negated—In MII mode, if TSEC <sub>n</sub> _RX_ER and TSEC <sub>n</sub> _RX_DV are asserted, the PHY has detected an error in the current frame. This signal is not used in the RGMII mode.
TSEC <sub>n</sub> _TX_CLK	I	Transmit clock in. In MII mode, TSEC <sub>n</sub> _TX_CLK is a continuous clock (2.5 or 25 MHz) that provides a timing reference for the TSEC <sub>n</sub> _TX_EN, TSEC <sub>n</sub> _TXD, and TSEC <sub>n</sub> _TX_ER signals. This signal is not used in the eTSEC RGMII mode.

Table 16-2. eTSEC Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description
TSEC <sub>n</sub> _TXD[3:0]	O	Transmit data out. DVIn MII mode, TSEC <sub>n</sub> _TXD[3:0] represent a nibble of data to be sent from the MAC to the PHY when TSEC <sub>n</sub> _TX_EN is asserted and have no meaning while TSEC <sub>n</sub> _TX_EN is negated. In RGMII mode, data bits 3:0 are transmitted on the rising edge of TSEC <sub>n</sub> _GTX_CLK, and data bits 7:4 are transmitted on the falling edge of TSEC <sub>n</sub> _GTX_CLK. Note that some of these signals are also used during reset to configure the eTSEC interface mode.
TSEC <sub>n</sub> _TX_EN	O	Transmit data valid. In MII mode, if TSEC <sub>n</sub> _TX_EN is asserted, the MAC is indicating that valid data is present on the MII's TSEC <sub>n</sub> _TXD signals. In RGMII mode, TSEC <sub>n</sub> _TX_EN becomes TX_CTL. TX_EN and TX_ERR are asserted on this signal on rising and falling edges of the TSEC <sub>n</sub> _GTX_CLK, respectively.
TSEC <sub>n</sub> _TX_ER	O	Transmit error. In MII mode, assertion of TSEC <sub>n</sub> _TX_ER for one or more clock cycles while TSEC <sub>n</sub> _TX_EN is asserted causes the PHY to transmit one or more illegal symbols. Asserting TSEC <sub>n</sub> _TX_ER has no effect while operating at 10 Mbps or while TSEC <sub>n</sub> _TX_EN is negated. This signal transitions synchronously with respect to TSEC <sub>n</sub> _TX_CLK. This signal is not used in the eTSEC and RGMII modes and is driven low.
TSEC_TMR_CLK	I	1588 clock in. External high precision timer reference clock input (chip external input pin).
TSEC_TMR_GCLK	O	1588 clock out. Phase aligned timer clock divider output (chip external output pin).
TSEC_TMR_TRIG1	I	1588 trigger in 1. External timer trigger input 1. This is an asynchronous general purpose input (chip external input pin).
TSEC_TMR_TRIG2	i	1588 trigger in 2. External timer trigger input 2. This is an asynchronous general purpose input (chip external input pin).
TSEC_TMR_PP1	O	1588 pulse out 1. Timer pulse per period 1. It is phase aligned with 1588 timer clock (chip external output pin)
TSEC_TMR_PP2	O	1588 pulse out 2. Timer pulse per period 2. It is phase aligned with 1588 timer clock (chip external output pin)
TSEC_TMR_PP3	O	1588 pulse out 3. Timer pulse per period 3. It is phase aligned with 1588 timer clock (chip external output pin)
TSEC_TMR_ALARM1	O	1588 timer alarm 1. Timer current time is equal to or greater than alarm time comparator register. User reprograms the TMR_ALARMn_H/L register to deactivate this output (chip external output pin)
TSEC_TMR_ALARM2	O	1588 timer alarm 2. Timer current time is equal to or greater than alarm time comparator register. User reprograms the TMR_ALARMn_H/L register to deactivate this output (chip external output pin)
TSEC <sub>n</sub> _TMR_RX_ESFD	I	Receive external start of frame delimiter
TSEC <sub>n</sub> _TMR_TX_ESFD	I	Transmit external start of frame delimiter

## 16.5 Memory Map/Register Definition

The eTSECs use a software model that is a superset of the PowerQUICC II Pro 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 16.6.7, “Buffer Descriptors.”](#)

### 16.5.1 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 16-3](#).

**Table 16-3. 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

### 16.5.2 Detailed Memory Map

The eTSEC memory mapped registers are accessed by reading and writing to an address comprised of the base address (specified in IMMRBAR as defined in [Chapter 3, “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 16-4](#) 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
- eTSEC2 starts at 0x2\_5000 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 16-4. Module Memory Map**

eTSEC1 Offset	Name <sup>1</sup>	Access <sup>2</sup>	Reset	Section/Page
<b>eTSEC 1—Block Base Address 0x2_4000 eTSEC 2—Block Base Address 0x2_5000</b>				
<b>eTSEC General Control and Status Registers</b>				
0x2_4000	TSEC_ID*—Controller ID register	R	0x0124_0106	<a href="#">16.5.3.1.1/16-21</a>
0x2_4004	TSEC_ID2*—Controller ID register	R	0x0030_00F0	<a href="#">16.5.3.1.2/16-22</a>
0x2_4008– 0x2_400C	Reserved	—	—	—
0x2_4010	IEVENT—Interrupt event register	w1c	0x0000_0000	<a href="#">16.5.3.1.3/16-22</a>
0x2_4014	IMASK—Interrupt mask register	R/W	0x0000_0000	<a href="#">16.5.3.1.4/16-26</a>
0x2_4018	EDIS—Error disabled register	R/W	0x0000_0000	<a href="#">16.5.3.1.5/16-28</a>
0x2_401C	Reserved	—	—	—
0x2_4020	ECNTRL—Ethernet control register	R/W	0x0000_0000	<a href="#">16.5.3.1.6/16-30</a>
0x2_4024	Reserved	—	—	—
0x2_4028	PTV—Pause time value register	R/W	0x0000_0000	<a href="#">16.5.3.1.7/16-31</a>
0x2_402C	DMACTRL—DMA control register	R/W	0x0000_0000	<a href="#">16.5.3.1.8/16-32</a>
0x2_4030	TBIPA—TBI PHY address register	R/W	0x0000_0000	<a href="#">16.5.3.1.9/16-33</a>
0x2_4034– 0x2_40FC	Reserved	—	—	—
<b>eTSEC Transmit Control and Status Registers</b>				
0x2_4100	TCTRL—Transmit control register	R/W	0x0000_0000	<a href="#">16.5.3.2.1/16-34</a>
0x2_4104	TSTAT—Transmit status register	w1c	0x0000_0000	<a href="#">16.5.3.2.2/16-36</a>
0x2_4108	DFVLAN*—Default VLAN control word	R/W	0x8100_0000	<a href="#">16.5.3.2.3/16-40</a>
0x2_410C	Reserved	—	—	—
0x2_4110	TXIC—Transmit interrupt coalescing register	R/W	0x0000_0000	<a href="#">16.5.3.2.4/16-41</a>
0x2_4114	TQUEUE*—Transmit queue control register	R/W	0x0000_8000	<a href="#">16.5.3.2.5/16-42</a>
0x2_4118– 0x2_413C	Reserved	—	—	—
0x2_4140	TR03WT*—TxBD Rings 0–3 round-robin weightings	R/W	0x0000_0000	<a href="#">16.5.3.2.6/16-42</a>

Table 16-4. Module Memory Map (continued)

eTSEC1 Offset	Name <sup>1</sup>	Access <sup>2</sup>	Reset	Section/Page
0x2_4144	TR47WT*—TxBD Rings 4–7 round-robin weightings	R/W	0x0000_0000	16.5.3.2.7/16-43
0x2_4148– 0x2_4180	Reserved	—	—	—
0x2_4184	TBPTR0—TxBD pointer for ring 0	R/W	0x0000_0000	16.5.3.2.8/16-44
0x2_4188	Reserved	—	—	—
0x2_418C	TBPTR1*—TxBD pointer for ring 1	R/W	0x0000_0000	16.5.3.2.8/16-44
0x2_4190	Reserved	—	—	—
0x2_4194	TBPTR2*—TxBD pointer for ring 2	R/W	0x0000_0000	16.5.3.2.8/16-44
0x2_4198	Reserved	—	—	—
0x2_419C	TBPTR3*—TxBD pointer for ring 3	R/W	0x0000_0000	16.5.3.2.8/16-44
0x2_41A0	Reserved	—	—	—
0x2_41A4	TBPTR4*—TxBD pointer for ring 4	R/W	0x0000_0000	16.5.3.2.8/16-44
0x2_41A8	Reserved	—	—	—
0x2_41AC	TBPTR5*—TxBD pointer for ring 5	R/W	0x0000_0000	16.5.3.2.8/16-44
0x2_41B0	Reserved	—	—	—
0x2_41B4	TBPTR6*—TxBD pointer for ring 6	R/W	0x0000_0000	16.5.3.2.8/16-44
0x2_41B8	Reserved	—	—	—
0x2_41BC	TBPTR7*—TxBD pointer for ring 7	R/W	0x0000_0000	16.5.3.2.8/16-44
0x2_41C0– 0x2_4200	Reserved	—	—	—
0x2_4204	TBASE0—TxBD base address of ring 0	R/W	0x0000_0000	16.5.3.2.9/16-45
0x2_4208	Reserved	—	—	—
0x2_420C	TBASE1*—TxBD base address of ring 1	R/W	0x0000_0000	16.5.3.2.9/16-45
0x2_4210	Reserved	—	—	—
0x2_4214	TBASE2*—TxBD base address of ring 2	R/W	0x0000_0000	16.5.3.2.9/16-45
0x2_4218	Reserved	—	—	—
0x2_421C	TBASE3*—TxBD base address of ring 3	R/W	0x0000_0000	16.5.3.2.9/16-45
0x2_4220	Reserved	—	—	—
0x2_4224	TBASE4*—TxBD base address of ring 4	R/W	0x0000_0000	16.5.3.2.9/16-45
0x2_4228	Reserved	—	—	—
0x2_422C	TBASE5*—TxBD base address of ring 5	R/W	0x0000_0000	16.5.3.2.9/16-45
0x2_4230	Reserved	—	—	—
0x2_4234	TBASE6*—TxBD base address of ring 6	R/W	0x0000_0000	16.5.3.2.9/16-45
0x2_4238	Reserved	—	—	—
0x2_423C	TBASE7*—TxBD base address of ring 7	R/W	0x0000_0000	16.5.3.2.9/16-45



Table 16-4. Module Memory Map (continued)

eTSEC1 Offset	Name <sup>1</sup>	Access <sup>2</sup>	Reset	Section/Page
0x2_4240–0x2_427C	Reserved	—	—	—
0x2_4280	TMR_TXTS1_ID* - Tx time stamp identification (set 1)	R/W	0x0000_0000	16.5.3.2.10/16-45
0x2_4284	TMR_TXTS2_ID* - Tx time stamp identification (set 2)	R/W	0x0000_0000	16.5.3.2.10/16-45
0x2_4288–0x2_42BC	Reserved	—	—	—
0x2_42C0	TMR_TXTS1_H* - Tx time stamp high (set 1)	R/W	0x0000_0000	16.5.3.2.11/16-46
0x2_42C4	TMR_TXTS1_L* - Tx time stamp high (set 1)	R/W	0x0000_0000	16.5.3.2.11/16-46
0x2_42C8	TMR_TXTS2_H* - Tx time stamp high (set 2)	R/W	0x0000_0000	16.5.3.2.11/16-46
0x2_42CC	TMR_TXTS2_L* - Tx time stamp high (set 2)	R/W	0x0000_0000	16.5.3.2.11/16-46
0x2_42D0–0x2_42FC	Reserved	—	—	—
<b>eTSEC Receive Control and Status Registers</b>				
0x2_4300	RCTRL—Receive control register	R/W	0x0000_0000	16.5.3.3.1/16-46
0x2_4304	RSTAT—Receive status register	w1c	0x0000_0000	16.5.3.3.2/16-48
0x2_4308–0x2_430C	Reserved	—	—	—
0x2_4310	RXIC—Receive interrupt coalescing register	R/W	0x0000_0000	16.5.3.3.3/16-50
0x2_4314	RQUEUE*—Receive queue control register.	R/W	0x0080_0080	16.5.3.3.4/16-51
0x2_4318–0x2_432C	Reserved	—	—	—
0x2_4330	RBIFX*—Receive bit field extract control register	R/W	0x0000_0000	16.5.3.3.5/16-52
0x2_4334	RQFAR*—Receive queue filing table address register	R/W	0x0000_0000	16.5.3.3.6/16-53
0x2_4338	RQFCR*—Receive queue filing table control register	R/W	0xn <sub>nnn</sub> _n <sub>nnn</sub>	16.5.3.3.7/16-54
0x2_433C	RQFPR*—Receive queue filing table property register	R/W	0xn <sub>nnn</sub> _n <sub>nnn</sub>	16.5.3.3.8/16-55
0x2_4340	MRBLR—Maximum receive buffer length register	R/W	0x0000_0000	16.5.3.3.9/16-59
0x2_4344–0x2_4380	Reserved	—	—	—
0x2_4384	BPTR0—RxB pointer for ring 0	R/W	0x0000_0000	16.5.3.3.10/16-60
0x2_4388	Reserved	—	—	—
0x2_438C	BPTR1*—RxB pointer for ring 1	R/W	0x0000_0000	16.5.3.3.10/16-60
0x2_4390	Reserved	—	—	—
0x2_4394	BPTR2*—RxB pointer for ring 2	R/W	0x0000_0000	16.5.3.3.10/16-60
0x2_4398	Reserved	—	—	—
0x2_439C	BPTR3*—RxB pointer for ring 3	R/W	0x0000_0000	16.5.3.3.10/16-60
0x2_43A0	Reserved	—	—	—

Table 16-4. Module Memory Map (continued)

eTSEC1 Offset	Name <sup>1</sup>	Access <sup>2</sup>	Reset	Section/Page
0x2_43A4	RBPTR4*—RxBd pointer for ring 4	R/W	0x0000_0000	16.5.3.3.10/16-60
0x2_43A8	Reserved	—	—	—
0x2_43AC	RBPTR5*—RxBd pointer for ring 5	R/W	0x0000_0000	16.5.3.3.10/16-60
0x2_43B0	Reserved	—	—	—
0x2_43B4	RBPTR6*—RxBd pointer for ring 6	R/W	0x0000_0000	16.5.3.3.10/16-60
0x2_43B8	Reserved	—	—	—
0x2_43BC	RBPTR7*—RxBd pointer for ring 7	R/W	0x0000_0000	16.5.3.3.10/16-60
0x2_43C0– 0x2_4400	Reserved	—	—	—
0x2_4404	RBASE0—RxBd base address of ring 0	R/W	0x0000_0000	16.5.3.3.11/16-60
0x2_4408	Reserved	—	—	—
0x2_440C	RBASE1*—RxBd base address of ring 1	R/W	0x0000_0000	16.5.3.3.11/16-60
0x2_4410	Reserved	—	—	—
0x2_4414	RBASE2*—RxBd base address of ring 2	R/W	0x0000_0000	16.5.3.3.11/16-60
0x2_4418	Reserved	—	—	—
0x2_441C	RBASE3*—RxBd base address of ring 3	R/W	0x0000_0000	16.5.3.3.11/16-60
0x2_4420	Reserved	—	—	—
0x2_4424	RBASE4*—RxBd base address of ring 4	R/W	0x0000_0000	16.5.3.3.11/16-60
0x2_4428	Reserved	—	—	—
0x2_442C	RBASE5*—RxBd base address of ring 5	R/W	0x0000_0000	16.5.3.3.11/16-60
0x2_4430	Reserved	—	—	—
0x2_4434	RBASE6*—RxBd base address of ring 6	R/W	0x0000_0000	16.5.3.3.11/16-60
0x2_4438	Reserved	—	—	—
0x2_443C	RBASE7*—RxBd base address of ring 7	R/W	0x0000_0000	16.5.3.3.11/16-60
0x2_4440– 0x2_44BC	Reserved	—	—	—
0x2_44C0	TMR_RXTS_H* - Rx timer time stamp register high	R/W	0x0000_0000	16.5.3.3.12/16-61
0x2_44C4	TMR_RXTS_L* - Rx timer time stamp register low	R/W	0x0000_0000	16.5.3.3.12/16-61
0x2_44C8– 0x2_44FC	Reserved	—	—	—
<b>eTSEC MAC Registers</b>				
0x2_4500	MACCFG1—MAC configuration register 1	R/W	0x0000_0000	16.5.3.5.1/16-64
0x2_4504	MACCFG2—MAC configuration register 2	R/W	0x0000_7000	16.5.3.5.2/16-66
0x2_4508	IPGIFG—Inter-packet/inter-frame gap register	R/W	0x4060_5060	16.5.3.5.3/16-68
0x2_450C	HAFDUP—Half-duplex control	R/W	0x00A1_F037	16.5.3.5.4/16-69

Table 16-4. Module Memory Map (continued)

eTSEC1 Offset	Name <sup>1</sup>	Access <sup>2</sup>	Reset	Section/Page
0x2_4510	MAXFRM—Maximum frame length	R/W	0x0000_0600	16.5.3.5.5/16-70
0x2_4514– 0x2_451C	Reserved	—	—	—
0x2_4520	MIIMCFG—MII management configuration	R/W	0x0000_0007	16.5.3.5.6/16-70
0x2_4524	MIIMCOM—MII management command	R/W	0x0000_0000	16.5.3.5.7/16-71
0x2_4528	MIIMADD—MII management address	R/W	0x0000_0000	16.5.3.5.8/16-72
0x2_452C	MIIMCON—MII management control	W	0x0000_0000	16.5.3.5.9/16-72
0x2_4530	MIIMSTAT—MII management status	R	0x0000_0000	16.5.3.5.10/16-73
0x2_4534	MIIMIND—MII management indicator	R	0x0000_0000	16.5.3.5.11/16-73
0x2_4538	Reserved	—	—	—
0x2_453C	IFSTAT—Interface status	R	0x0000_0000	16.5.3.5.12/16-74
0x2_4540	MACSTNADDR1—MAC station address register 1	R/W	0x0000_0000	16.5.3.5.13/16-74
0x2_4544	MACSTNADDR2—MAC station address register 2	R/W	0x0000_0000	16.5.3.5.14/16-75
0x2_4548	MAC01ADDR1*—MAC exact match address 1, part 1	R/W	0x0000_0000	16.5.3.5.15/16-75 16.5.3.5.16/16-76
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 16-4. Module Memory Map (continued)

eTSEC1 Offset	Name <sup>1</sup>	Access <sup>2</sup>	Reset	Section/Page
0x2_4570	MAC06ADDR1*—MAC exact match address 6, part 1	R/W	0x0000_0000	16.5.3.5.15/16-75 16.5.3.5.16/16-76
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	—	—	—
<b>eTSEC Transmit and Receive Counters</b>				
0x2_4680	TR64—Transmit and receive 64-byte frame counter	R/W	0x0000_0000	16.5.3.6.1/16-78
0x2_4684	TR127—Transmit and receive 65- to 127-byte frame counter	R/W	0x0000_0000	16.5.3.6.2/16-78
0x2_4688	TR255—Transmit and receive 128- to 255-byte frame counter	R/W	0x0000_0000	16.5.3.6.3/16-79
0x2_468C	TR511—Transmit and receive 256- to 511-byte frame counter	R/W	0x0000_0000	16.5.3.6.4/16-79
0x2_4690	TR1K—Transmit and receive 512- to 1023-byte frame counter	R/W	0x0000_0000	16.5.3.6.5/16-80
0x2_4694	TRMAX—Transmit and receive 1024- to 1518-byte frame counter	R/W	0x0000_0000	16.5.3.6.6/16-80
0x2_4698	TRMGV—Transmit and receive 1519- to 1522-byte good VLAN frame count	R/W	0x0000_0000	16.5.3.6.7/16-81
<b>eTSEC Receive Counters</b>				
0x2_469C	RBYT—Receive byte counter	R/W	0x0000_0000	16.5.3.6.8/16-81
0x2_46A0	RPKT—Receive packet counter	R/W	0x0000_0000	16.5.3.6.9/16-81
0x2_46A4	RFCS—Receive FCS error counter	R/W	0x0000_0000	16.5.3.6.10/16-82
0x2_46A8	RMCA—Receive multicast packet counter	R/W	0x0000_0000	16.5.3.6.11/16-82

Table 16-4. Module Memory Map (continued)

eTSEC1 Offset	Name <sup>1</sup>	Access <sup>2</sup>	Reset	Section/Page
0x2_46AC	RBCA—Receive broadcast packet counter	R/W	0x0000_0000	16.5.3.6.12/16-83
0x2_46B0	RXCF—Receive control frame packet counter	R/W	0x0000_0000	16.5.3.6.13/16-83
0x2_46B4	RXPF—Receive PAUSE frame packet counter	R/W	0x0000_0000	16.5.3.6.14/16-84
0x2_46B8	RXUO—Receive unknown OP code counter	R/W	0x0000_0000	16.5.3.6.15/16-84
0x2_46BC	RALN—Receive alignment error counter	R/W	0x0000_0000	16.5.3.6.16/16-85
0x2_46C0	RFLR—Receive frame length error counter	R/W	0x0000_0000	16.5.3.6.17/16-85
0x2_46C4	RCDE—Receive code error counter	R/W	0x0000_0000	16.5.3.6.18/16-86
0x2_46C8	RCSE—Receive carrier sense error counter	R/W	0x0000_0000	16.5.3.6.19/16-86
0x2_46CC	RUND—Receive undersize packet counter	R/W	0x0000_0000	16.5.3.6.20/16-87
0x2_46D0	ROVR—Receive oversize packet counter	R/W	0x0000_0000	16.5.3.6.21/16-87
0x2_46D4	RFRG—Receive fragments counter	R/W	0x0000_0000	16.5.3.6.22/16-88
0x2_46D8	RJBR—Receive jabber counter	R/W	0x0000_0000	16.5.3.6.23/16-88
0x2_46DC	RDRP—Receive drop counter	R/W	0x0000_0000	16.5.3.6.24/16-89
<b>eTSEC Transmit Counters</b>				
0x2_46E0	TBYT—Transmit byte counter	R/W	0x0000_0000	16.5.3.6.25/16-89
0x2_46E4	TPKT—Transmit packet counter	R/W	0x0000_0000	16.5.3.6.26/16-90
0x2_46E8	TMCA—Transmit multicast packet counter	R/W	0x0000_0000	16.5.3.6.27/16-90
0x2_46EC	TBCA—Transmit broadcast packet counter	R/W	0x0000_0000	16.5.3.6.28/16-91
0x2_46F0	TXPF—Transmit PAUSE control frame counter	R/W	0x0000_0000	16.5.3.6.29/16-91
0x2_46F4	TDFR—Transmit deferral packet counter	R/W	0x0000_0000	16.5.3.6.30/16-92
0x2_46F8	TEDF—Transmit excessive deferral packet counter	R/W	0x0000_0000	16.5.3.6.31/16-92
0x2_46FC	TSCL—Transmit single collision packet counter	R/W	0x0000_0000	16.5.3.6.32/16-93
0x2_4700	TMCL—Transmit multiple collision packet counter	R/W	0x0000_0000	16.5.3.6.33/16-93
0x2_4704	TLCL—Transmit late collision packet counter	R/W	0x0000_0000	16.5.3.6.34/16-94
0x2_4708	TXCL—Transmit excessive collision packet counter	R/W	0x0000_0000	16.5.3.6.35/16-94
0x2_470C	TNCL—Transmit total collision counter	R/W	0x0000_0000	16.5.3.6.36/16-95
0x2_4710	Reserved	—	—	—
0x2_4714	TDRP—Transmit drop frame counter	R/W	0x0000_0000	16.5.3.6.37/16-95
0x2_4718	TJBR—Transmit jabber frame counter	R/W	0x0000_0000	16.5.3.6.38/16-96
0x2_471C	TFCS—Transmit FCS error counter	R/W	0x0000_0000	16.5.3.6.39/16-96
0x2_4720	TXCF—Transmit control frame counter	R/W	0x0000_0000	16.5.3.6.40/16-97
0x2_4724	TOVR—Transmit oversize frame counter	R/W	0x0000_0000	16.5.3.6.41/16-97
0x2_4728	TUND—Transmit undersize frame counter	R/W	0x0000_0000	16.5.3.6.42/16-98
0x2_472C	TFRG—Transmit fragments frame counter	R/W	0x0000_0000	16.5.3.6.43/16-98

Table 16-4. Module Memory Map (continued)

eTSEC1 Offset	Name <sup>1</sup>	Access <sup>2</sup>	Reset	Section/Page
<b>eTSEC Counter Control and TOE Statistics Registers</b>				
0x2_4730	CAR1—Carry register one register <sup>3</sup>	w1c	0x0000_0000	16.5.3.6.44/16-99
0x2_4734	CAR2—Carry register two register <sup>3</sup>	w1c	0x0000_0000	16.5.3.6.45/16-100
0x2_4738	CAM1—Carry register one mask register	R/W	0xFE03_FFFF	16.5.3.6.46/16-101
0x2_473C	CAM2—Carry register two mask register	R/W	0x000F_FFFD	16.5.3.6.47/16-103
0x2_4740	RREJ*—Receive filer rejected packet counter	R/W	0x0000_0000	16.5.3.6.48/16-104
0x2_4744– 0x2_47FC	Reserved	—	—	—
<b>Hash Function Registers</b>				
0x2_4800	IGADDR0—Individual/group address register 0	R/W	0x0000_0000	16.5.3.7.1/16-105
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– 0x2_487C	Reserved	—	—	—
0x2_4880	GADDR0—Group address register 0	R/W	0x0000_0000	16.5.3.7.2/16-105
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– 0x2_4AFC	Reserved	—	—	—
<b>eTSEC DMA Attribute Registers</b>				
0x2_4B00– 0x2_4BF4	Reserved	—	—	—
0x2_4BF8	ATTR—Attribute register	R/W	0x0000_0000	16.5.3.8.1/16-106
0x2_4BFC	ATTRELI*—Attribute extract length and extract index register	R/W	0x0000_0000	16.5.3.8.2/16-106

Table 16-4. Module Memory Map (continued)

eTSEC1 Offset	Name <sup>1</sup>	Access <sup>2</sup>	Reset	Section/Page
<b>eTSEC Lossless Flow Control Registers</b>				
0x2_4C00	RQPRM0*—Receive Queue Parameters register 0	R/W	0x0000_0000	16.5.3.9.1/16-107
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– 0x2_4C40	Reserved	—	—	—
0x2_4C44	RFBPTR0*—Last Free RxBD pointer for ring 0	R/W	0x0000_0000	16.5.3.9.2/16-108
0x2_4C48	Reserved	—	—	—
0x2_4C4C	RFBPTR1*—Last Free RxBD pointer for ring 1	R/W	0x0000_0000	16.5.3.9.2/16-108
0x2_4C50	Reserved	—	—	—
0x2_4C54	RFBPTR2*—Last Free RxBD pointer for ring 2	R/W	0x0000_0000	16.5.3.9.2/16-108
0x2_4C58	Reserved	—	—	—
0x2_4C5C	RFBPTR3*—Last Free RxBD pointer for ring 3	R/W	0x0000_0000	16.5.3.9.2/16-108
0x2_4C60	Reserved	—	—	—
0x2_4C64	RFBPTR4*—Last Free RxBD pointer for ring 4	R/W	0x0000_0000	16.5.3.9.2/16-108
0x2_4C68	Reserved	—	—	—
0x2_4C6C	RFBPTR5*—Last Free RxBD pointer for ring 5	R/W	0x0000_0000	16.5.3.9.2/16-108
0x2_4C70	Reserved	—	—	—
0x2_4C74	RFBPTR6*—Last Free RxBD pointer for ring 6	R/W	0x0000_0000	16.5.3.9.2/16-108
0x2_4C78	Reserved	—	—	—
0x2_4C7C	RFBPTR7*—Last Free RxBD pointer for ring 7	R/W	0x0000_0000	16.5.3.9.2/16-108
<b>eTSEC Future Expansion Space</b>				
0x2_4CC0– 0x2_4D94	Reserved	—	—	—
<b>eTSEC IEEE 1588 Registers</b>				
0x2_4E00	TMR_CTRL* - Timer control register	R/W	0x0001_0001	16.5.3.10.1/16-109
0x2_4E04	TMR_TEVENT* - time stamp event register	w1c	0x0000_0000	16.5.3.10.2/16-111
0x2_4E08	TMR_TEMASK* - Timer event mask register	R/W	0x0000_0000	16.5.3.10.3/16-113
0x2_4E0C	TMR_PEVENT* - time stamp event register	R/W	0x0000_0000	16.5.3.10.4/16-113
0x2_4E10	TMR_PEMASK* - Timer event mask register	R/W	0x0000_0000	16.5.3.10.5/16-114

Table 16-4. Module Memory Map (continued)

eTSEC1 Offset	Name <sup>1</sup>	Access <sup>2</sup>	Reset	Section/Page
0x2_4E14	TMR_STAT* - time stamp status register	R/W	0x0000_0000	16.5.3.10.6/16-115
0x2_4E18	TMR_CNT_H* - timer counter high register	R/W	0x0000_0000	16.5.3.10.7/16-115
0x2_4E1C	TMR_CNT_L* - timer counter low register	R/W	0x0000_0000	16.5.3.10.7/16-115
0x2_4E20	TMR_ADD* - Timer drift compensation addend register	R/W	0x0000_0000	16.5.3.10.8/16-116
0x2_4E24	TMR_ACC* - Timer accumulator register	R/W	0x0000_0000	16.5.3.10.9/16-117
0x2_4E28	TMR_PRSC* -Timer prescale	R/W	0x0000_0002	16.5.3.10.10/16-117
0x2_4E2C	Reserved	—	—	—
0x2_4E30	TMROFF_H* - Timer offset high	R/W	0x0000_0000	16.5.3.10.11/16-118
0x2_4E34	TMROFF_L* - Timer offset low	R/W	0x0000_0000	16.5.3.10.11/16-118
0x2_4E40	TMR_ALARM1_H* - Timer alarm 1 high register	R/W	0xFFFF_FFFF	16.5.3.10.12/16-118
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	16.5.3.10.13/16-119
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	16.5.3.10.14/16-120
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	—	—	—
<b>Other eTSECs</b>				
0x2_5000– 0x2_5FFF	eTSEC2 REGISTERS <sup>4</sup>			

<sup>1</sup> Registers denoted \* are new to the enhanced TSEC and not supported by PowerQUICC II Pro TSECs.

<sup>2</sup> Key: R = read only, WO = write only, R/W = read and write, LH = latches high, SC = self-clearing.

<sup>3</sup> Cleared on read.

<sup>4</sup> eTSEC2 has the same memory-mapped registers that are described for eTSEC1 from 0x2\_4000 to 0x2\_4FFF, except the offsets are from 0x2\_5000 to 0x2\_5FFF.



## 16.5.3 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.

### 16.5.3.1 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.

#### 16.5.3.1.1 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.

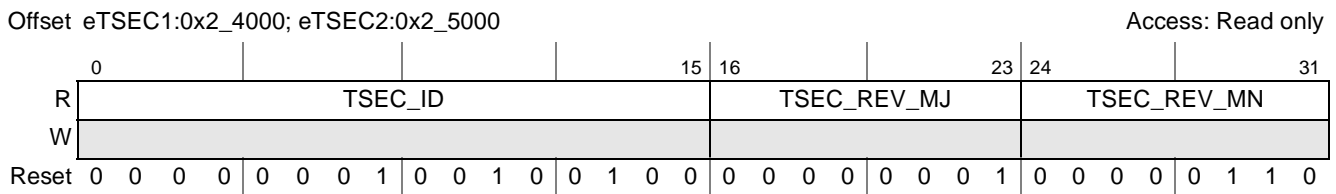


Figure 16-2. TSEC\_ID Register

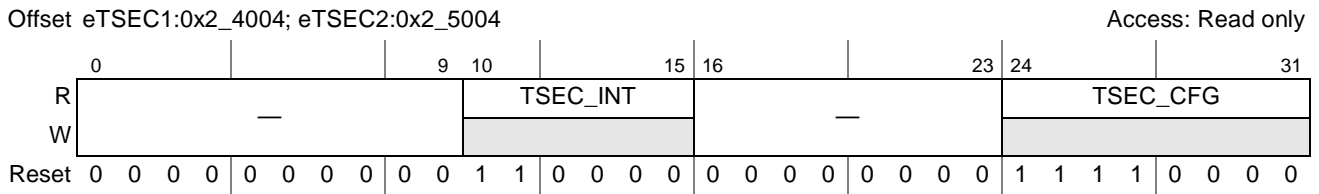
Table 16-5 describes the fields of the TSEC\_ID register.

Table 16-5. TSEC\_ID Field Descriptions

Bits	Name	Description
0–15	TSEC_ID	Value identifying the eTSEC (10/100/1000 Ethernet MAC). 0124 Unique identifier for eTSEC with 8 Rx and 8 Tx BD rings. 0800 Unique identifier for GMAC1 with 8 Rx and 8 Tx BD rings. 0810 Unique identifier for GMAC2 with 8 Rx and 8 Tx BD rings.
16–23	TSEC_REV_MJ	Value identifies the major revision of the eTSEC.  01 Silicon Rev 2.1
24–31	TSEC_REV_MN	Value identifies the minor revision of the eTSEC.  06 Silicon Rev 2.1

### 16.5.3.1.2 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.



**Figure 16-3. TSEC\_ID2 Register**

Table 16-6 describes the fields of the TSEC\_ID2 register.

**Table 16-6. TSEC\_ID2 Field Descriptions**

Bits	Name	Description
0–9	—	Reserved
10–15	TSEC_INT	Interface mode support. See Table 16-7 for settings.
16–23	—	Reserved
24–31	TSEC_CFG	Value identifies configuration options of the eTSEC. 00 eTSEC multiple ring, Rx TOE, Filer and Tx TOE supports are off F0 eTSEC multiple ring, Rx TOE, Filer and Tx TOE supports are on 30 eTSEC multiple ring support is OFF and Rx TOE, Filer and Tx TOE supports are on 50 eTSEC multiple ring and filer supports are OFF and Rx TOE and Tx TOE supports are on

Table 16-7 describes the field settings for TSEC\_ID2[TSEC\_INT].

**Table 16-7. TSEC\_ID2[TSEC\_INT] Field Settings**

Bit	Mode
10	0 Ethernet mode not supported 1 Ethernet mode supported
11–13	Reserved
14	0 Can be configured to run in Ethernet normal/full mode 1 Ethernet normal/full mode off
15	0 Can be configured to run in Ethernet reduced mode 1 Ethernet reduced mode off

### 16.5.3.1.3 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 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.
  - 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 16-4 describes the definition for the IEVENT register.

Offset eTSEC1:0x2\_4010; eTSEC2: 0x2\_5010 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	20	21	22	23	24	25	26	27	28	29	30	31
R	RXB	—	MMRD	MMWR	GRSC	RXF	—	FGPI	FIR	FIQ	DPE	PERR		
W	w1c	—	w1c	w1c	w1c	w1c	—	w1c	w1c	w1c	w1c	w1c		
Reset	All zeros													

Figure 16-4. IEVENT Register Definition

Table 16-8 describes the fields of the IEVENT register.

**Table 16-8. 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. 0 Excessive frame not received. 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. 0 Control frame not received. 1 Control frame received.
2	BSY	Busy condition interrupt. Indicates that a frame was received and discarded due to a lack of buffers. 0 No frame received and discarded. 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. 0 No system bus error occurred. 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. 0 MIB count not exceeding its register size. 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. <ul style="list-style-type: none"> <li>• A graceful stop, which was initiated by setting DMACTRL[GTS], is now complete.</li> <li>• A transmission of a flow control PAUSE frame, which was initiated by setting TCTRL[TFC_PAUSE], is now complete.</li> </ul> 0 No graceful stop interrupt. 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. 0 Transmitted frame length not exceeding maximum frame length. 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. 0 Control frame not transmitted. 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). 0 No transmit channel error occurred. 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. 0 No transmit buffer descriptor updated. 1 Transmit buffer descriptor updated.

Table 16-8. 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. 0 No frame transmitted/TxBD not updated. 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. 0 No late collision occurred. 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. 0 Successive transmission collisions do not exceed maximum. 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. 0 Transmit FIFO not underrun. 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. 0 Receive buffer descriptor not updated. 1 Receiver buffer descriptor updated.
17–20	—	Reserved
21	MMRD	MII management read completion 0 MII management read not issued or in process. 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 0 MII management write not issued or in process. 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. 0 Graceful stop not completed. 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. 0 Frame not received. 1 Frame received.
25–26	—	Reserved

Table 16-8. 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. 0 No filer generated interrupt has occurred. 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. 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. 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. 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. 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. 0 No parity errors detected. 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. 0 Received frame parsed successfully. 1 Received frame parse revealed header inconsistencies.

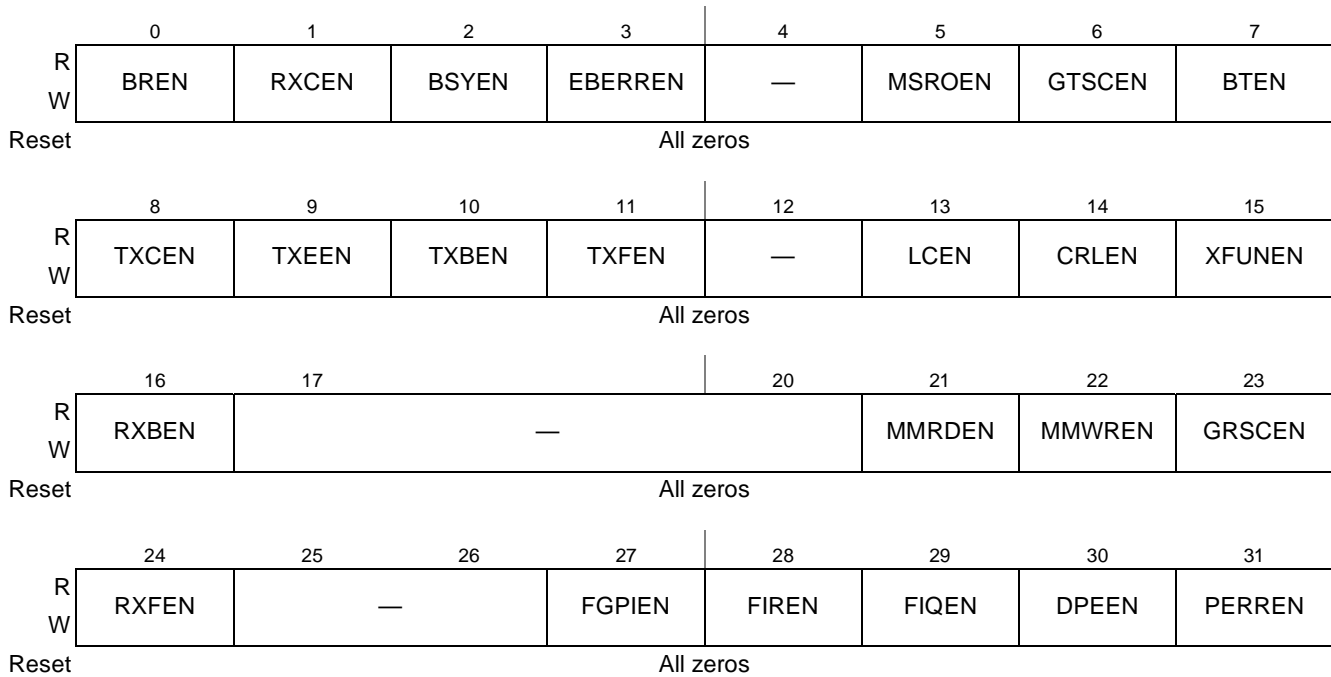
#### 16.5.3.1.4 Interrupt Mask Register (IMASK)

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 16-5 describes the IMASK register.

Offset eTSEC1:0x2\_4014; eTSEC2:0x2\_5014

Access: Read/Write



**Figure 16-5. IMASK Register Definition**

Table 16-9 describes the fields of the IMASK register.

**Table 16-9. 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 16-9. IMASK Field Descriptions (continued)**

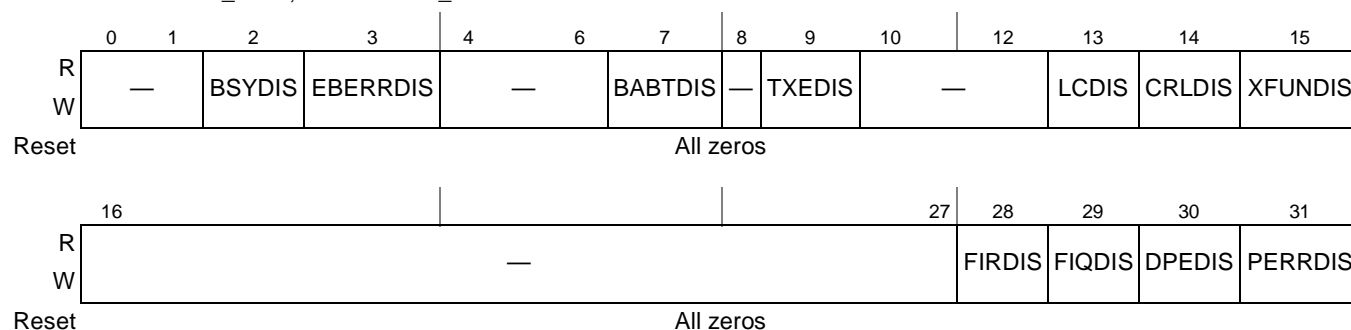
Bits	Name	Description
13	LCEN	Late collision enable
14	CRLEN	Collision retry limit enable
15	XFUNEN	Transmit FIFO underrun enable
16	RXBEN	Receive buffer interrupt enable
17–20	—	Reserved
21	MMRDEN	MII management read completion interrupt enable
22	MMWREN	MII management write completion interrupt enable
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

**16.5.3.1.5 Error Disabled Register (EDIS)**

Figure 16-6 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; eTSEC2:0x2\_5018

Access: Read/Write



**Figure 16-6. EDIS Register Definition**



Table 16-10 describes the fields of the EDIS register.

**Table 16-10. EDIS Field Descriptions**

Bits	Name	Description
0–1	—	Reserved
2	BSYDIS	Busy disable. 0 Allow eTSEC to report IEVENT[BSY] status and halt buffer descriptor queue if BSY condition occurs. 1 Do not set IEVENT[BSY] and do not halt buffer descriptor queue if BSY condition occurs.
3	EBERRDIS	Ethernet controller bus error disable. 0 Allow eTSEC to report IEVENT[EBERR] status and halt buffer descriptor queue if EBERR condition occurs. 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. 0 Allow eTSEC to report IEVENT[BABT] status and set the buffer descriptor TR field. 1 Do not set IEVENT[BABT] nor the buffer descriptor TR field.
8	—	Reserved
9	TXEDIS	Transmit error disable. 0 Allow eTSEC to report IEVENT[TXE] status. 1 Do not set IEVENT[TXE] if TXE condition occurs.
10–12	—	Reserved
13	LCDIS	Late collision disable. 0 Allow eTSEC to report IEVENT[LC] status, set the buffer descriptor LC field, and halt buffer descriptor queue if LC condition occurs. 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. 0 Allow eTSEC to report IEVENT[CRL] status, set the buffer descriptor RL field, and halt buffer descriptor queue if CRL condition occurs. 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. 0 Allow eTSEC to report IEVENT[XFUN] status, set the buffer descriptor UN field, and halt buffer descriptor queue if XFUN condition occurs. 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. 0 Allow eTSEC to report IEVENT[FIR] status. 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. 0 Allow eTSEC to report IEVENT[FIQ] status. 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 16-10. EDIS Field Descriptions (continued)**

Bits	Name	Description
30	DPEDIS	Data parity error disable. 0 Allow eTSEC to report IEVENT[DPE] status. 1 Do not set IEVENT[DPE] if a parity error occurs in eTSEC's FIFO or filter arrays.
31	PERRDIS	Receive frame parse error disable. 0 Allow eTSEC to report IEVENT[PERR] status. 1 Do not set IEVENT[PERR] if a parse error occurs on a received frame.

### 16.5.3.1.6 Ethernet Control Register (ECNTRL)

ECNTRL is a register writable by the user to reset, configure, and initialize the eTSEC. Note that the FIFM, GMIIM, RPM fields are read-only, having been set after sampling signals at power-on-reset. For more information, see TSEC mode in 4.3.2.2, “Reset Configuration Word High Register (RCWHR).”

Figure 16-7 describes the definition for the ECNTRL register.

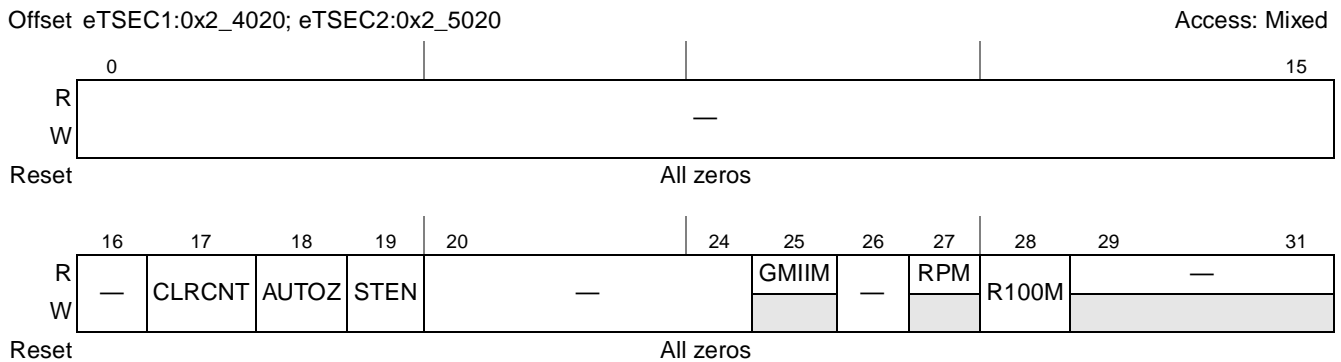
**Figure 16-7. ECNTRL Register Definition**

Table 16-11 describes the fields of the ECNTRL register.

**Table 16-11. ECNTRL Field Descriptions**

Bits	Name	Description
0–16	—	Reserved
17	CLRCNT	Clear all statistics counters and carry registers. 0 Allow MIB counters to continue to increment and keep any overflow indicators. 1 Reset all MIB counters and CAR1 and CAR2. This bit is self-resetting.
18	AUTOZ	Automatically zero MIB counter values and carry registers. 0 The user must write the addressed counter zero after a host read. 1 The addressed counter value is automatically cleared to zero after a host read. This is a steady state signal and must be set prior to enabling the Ethernet controller and must not be changed without proper care.

**Table 16-11. ECNTRL Field Descriptions (continued)**

Bits	Name	Description
19	STEN	MIB counter statistics enabled. 0 Statistics not enabled 1 Enables internal counters to update 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 RGMII interface is expected to be connected. If cleared, a PHY with an MII 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 <a href="#">Section 4.3.2, “Reset Configuration Words.”</a> 0 MII mode interface expected 1 RGMII mode interface expected
26	—	Reserved
27	RPM	Reduced-pin mode for Gigabit interfaces. If this bit is set, a reduced-pin interface is expected on Ethernet interfaces. This register can be pin-configured at reset to 0 or 1. 0 MII in non-reduced-pin mode configuration 1 RGMII reduced-pin mode
28	R100M	RGMII 100 mode. This bit is ignored unless RPM are set and MACCFG2[I/F Mode] is assigned to 10/100 (01). 0 RGMII is in 10 Mbps mode 1 RGMII is in 100 Mbps mode
29–31	—	Reserved

[Table 16-12](#) lists the different interface configurations indicated by registers, ECNTRL and MACCFG2.

**Table 16-12. eTSEC Interface Configurations**

Interface Mode	ECNTRL Field			MACCFG2 Field
	GMIIM	RPM	R100M	I/F Mode
RGMII 1 Gbps	1	1	0	10
RGMII 100 Mbps	1	1	1	01
RGMII 10 Mbps	1	1	0	01
MII 10/100 Mbps	0	0	0	01

### 16.5.3.1.7 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 16.6.2.8, “Flow Control,”](#) for additional details. [Figure 16-8](#) describes the definition for the PTV register.

Offset eTSEC1:0x2\_4028; eTSEC2:0x2\_5028

Access: Read/Write

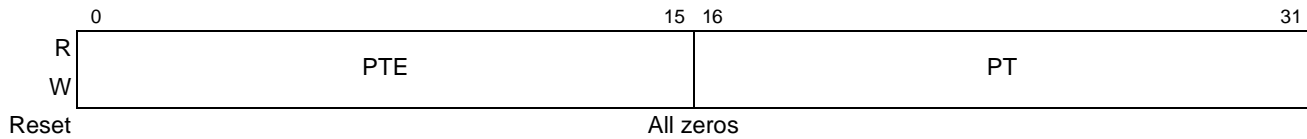


Figure 16-8. PTV Register Definition

Table 16-13 describes the fields of the PTV register.

Table 16-13. 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 16.6.2.8, “Flow Control,” for more information.

### 16.5.3.1.8 DMA Control Register (DMACTRL)

DMACTRL is writable by the user to configure the DMA block. Figure 16-9 describes the definition for the DMACTRL register.

Offset eTSEC1:0x2\_402C; eTSEC2:0x2\_502C

Access: Read/Write

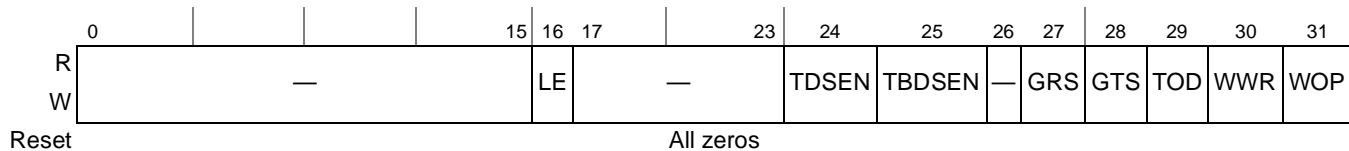


Figure 16-9. DMACTRL Register

Table 16-14 describes the fields of the DMACTRL register.

Table 16-14. 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. 0 RxBDs and TxBDs are interpreted with big-endian byte ordering, as shown in Section 16.6.7.1, “Data Buffer Descriptors.” 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
24	TDSSEN	Tx Data snoop enable. 0 Disables snooping of all transmit frames from memory. 1 Enables snooping of all transmit frames from memory.

Table 16-14. DMACTRL Field Descriptions (continued)

Bits	Name	Description
25	TBDSEN	TxBD snoop enable. 0 Disables snooping of all transmit BD memory accesses. 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. 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. 0 eTSEC scans input data stream for valid frame. 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. 0 Controller continues. 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[TXSCHE] = 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. 0 eTSEC continues waiting for the TxBD ring 0 poll timer to expire. 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. 0 Do not wait for acknowledgement from system for BD writes before setting IEVENT bits. 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[TXSCHE] = 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]. 0 Poll TxBD on ring 0 every 512 serial clocks. 1 Do not poll, but wait for TSTAT[THLT] to be cleared by the user.

### 16.5.3.1.9 TBI PHY Address Register (TBIPA)

TBI PHY address (TBIPA) register, as shown in [Figure 16-10](#), is writable by the users to assign a physical address to the MII management configuration.

Offset eTSEC1:0x2\_4030; eTSEC2:0x2\_5030

Access: Mixed

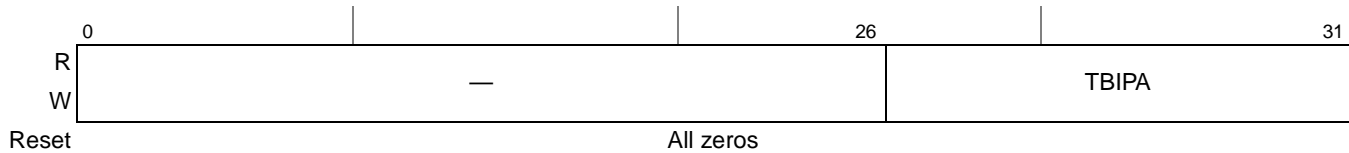


Figure 16-10. TBIPA Register Definition

Table 16-15 describes the fields of the TBIPA register.

Table 16-15. TBIPA Field Descriptions

Bits	Name	Description
0–26	—	Reserved
27–31	TBIPA	This field holds the PHY address of the MII management interface. <b>Note:</b> The value of 0 is reserved. External PHY cannot have the address 0.

### 16.5.3.2 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.

#### 16.5.3.2.1 Transmit Control Register (TCTRL)

This register is writable by the user to configure the transmit block. Figure 16-11 describes the TCTRL register.

Offset eTSEC1:0x2\_4100; eTSEC2:0x2\_5100

Access: Mixed



Figure 16-11. TCTRL Register Definition

Table 16-16 describes the fields of the TCTRL register.

Table 16-16. 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 16.6.3.2, “Transmit Path Off-Load and Tx PTP Packet Parsing.” 0 IP header checksum generation is disabled even if enabled in a transmit frame control block. 1 IP header checksum generation is performed for IPv4 headers as determined by the settings in the current transmit frame control block.

Table 16-16. TCTRL Field Descriptions (continued)

Bits	Name	Description
18	TUCSEN	TCP/UDP header checksum generation enable. When set, the eTSEC offloads TCP or UDP header checksum generation. See <a href="#">Section 16.6.3.2, “Transmit Path Off-Load and Tx PTP Packet Parsing.”</a> 0 TCP or UDP header checksum generation is disabled even if enabled in a transmit frame control block. 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. 0 Do not insert a VLAN tag into the frame. 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. 0 Disable back pressure 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. 0 Pause duration complete. 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. 0 No request for Tx PAUSE frame pending or transmission complete. 1 Software request for Tx PAUSE frame pending.
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. 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. 01 Priority scheduling mode. Frames from enabled TxBD rings are serviced in ascending ring index order. 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 $WT_n \times 64$ 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. 11 Reserved
31	—	Reserved

### 16.5.3.2.2 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 16-12 describes the TSTAT register.

Offset eTSEC1:0x2\_4104; eTSEC2:0x2\_5104 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 16-12. TSTAT Register Definition**



Table 16-17 describes the fields of the TSTAT register.

**Table 16-17. TSTAT Field Descriptions**

Bits	Name	Description
0	THLT0	<p>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.</p> <p>Repeatable error conditions that cause halt include: Bus error:</p> <ul style="list-style-type: none"> <li>• Invalid BD or data address</li> <li>• Uncorrectable error on BD or data read</li> </ul> <p>TxBD programming errors:</p> <ul style="list-style-type: none"> <li>• Ready = 1 and length = 0</li> </ul>
1	THLT1	<p>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.</p> <p>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.</p> <p>Repeatable error conditions that cause halt include: Bus error:</p> <ul style="list-style-type: none"> <li>• Invalid BD or data address</li> <li>• Uncorrectable error on BD or data read</li> </ul> <p>TxBD programming errors:</p> <ul style="list-style-type: none"> <li>• Ready = 1 and length = 0</li> </ul>
2	THLT2	<p>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.</p> <p>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.</p> <p>Repeatable error conditions that cause halt include: Bus error:</p> <ul style="list-style-type: none"> <li>• Invalid BD or data address</li> <li>• Uncorrectable error on BD or data read</li> </ul> <p>TxBD programming errors:</p> <ul style="list-style-type: none"> <li>• Ready=1 and length=0</li> </ul>

Table 16-17. TSTAT Field Descriptions (continued)

Bits	Name	Description
3	THLT3	<p>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.</p> <p>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.</p> <p>Repeatable error conditions that cause halt include: Bus error:</p> <ul style="list-style-type: none"> <li>• Invalid BD or data address</li> <li>• Uncorrectable error on BD or data read</li> </ul> <p>TxBD programming errors:</p> <ul style="list-style-type: none"> <li>• Ready=1 and length=0</li> </ul>
4	THLT4	<p>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.</p> <p>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.</p> <p>Repeatable error conditions that cause halt include: Bus error:</p> <ul style="list-style-type: none"> <li>• Invalid BD or data address</li> <li>• Uncorrectable error on BD or data read</li> </ul> <p>TxBD programming errors:</p> <ul style="list-style-type: none"> <li>• Ready=1 and length=0</li> </ul>
5	THLT5	<p>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.</p> <p>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.</p> <p>Repeatable error conditions that cause halt include: Bus error:</p> <ul style="list-style-type: none"> <li>• Invalid BD or data address</li> <li>• Uncorrectable error on BD or data read</li> </ul> <p>TxBD programming errors:</p> <ul style="list-style-type: none"> <li>• Ready=1 and length=0</li> </ul>

Table 16-17. TSTAT Field Descriptions (continued)

Bits	Name	Description
6	THLT6	<p>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.</p> <p>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.</p> <p>Repeatable error conditions that cause halt include: Bus error:</p> <ul style="list-style-type: none"> <li>• Invalid BD or data address</li> <li>• Uncorrectable error on BD or data read</li> </ul> <p>TxBD programming errors:</p> <ul style="list-style-type: none"> <li>• Ready=1 and length=0</li> </ul>
7	THLT7	<p>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.</p> <p>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.</p> <p>Repeatable error conditions that cause halt include: Bus error:</p> <ul style="list-style-type: none"> <li>• Invalid BD or data address</li> <li>• Uncorrectable error on BD or data read</li> </ul> <p>TxBD programming errors:</p> <ul style="list-style-type: none"> <li>• Ready=1 and length=0</li> </ul>
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 16-17. 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

### 16.5.3.2.3 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 16-13 describes the DFVLAN register.

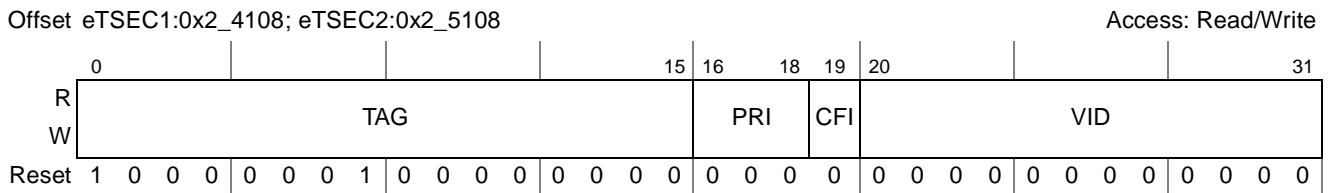


Figure 16-13. DFVLAN Register Definition

Table 16-18 describes the fields of the DFVLAN register.

Table 16-18. 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. 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.

### 16.5.3.2.4 Transmit Interrupt Coalescing Register (TXIC)

The TXIC register enables and configures the operational parameters for interrupt coalescing associated with transmitted frames. Figure 16-14 describes the definition for the TXIC register.

Offset eTSEC1:0x2\_4110; eTSEC2:0x2\_5110

Access: Read/Write

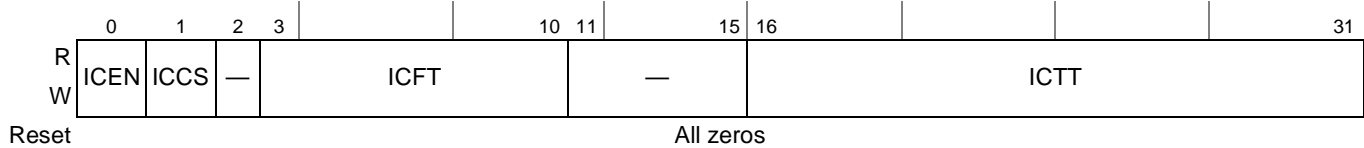


Figure 16-14. TXIC Register Definition

Table 16-19 describes the fields of the TXIC register.

Table 16-19. TXIC Field Descriptions

Bits	Name	Description
0	ICEN	Interrupt coalescing enable 0 Interrupt coalescing is disabled. Interrupts are raised as they are received. 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. 0 The coalescing timer advances count every 64 eTSEC Tx interface clocks (TSECn_GTX_CLK). 1 The coalescing timer advances count every 64 system clocks. 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[!] 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.

### 16.5.3.2.5 Transmit Queue Control Register (TQUEUE)

The TQUEUE register, shown in Figure 16-15, selectively enables each of the TxBD rings 0–7. By default, TxBD ring 0 is enabled.

Offset eTSEC1:0x2\_4114; eTSEC2:0x2\_5114

Access: Read/Write

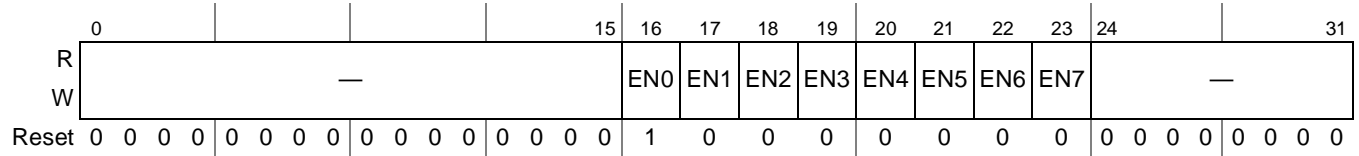


Figure 16-15. TQUEUE Register Definition

Table 16-20 describes the TQUEUE register.

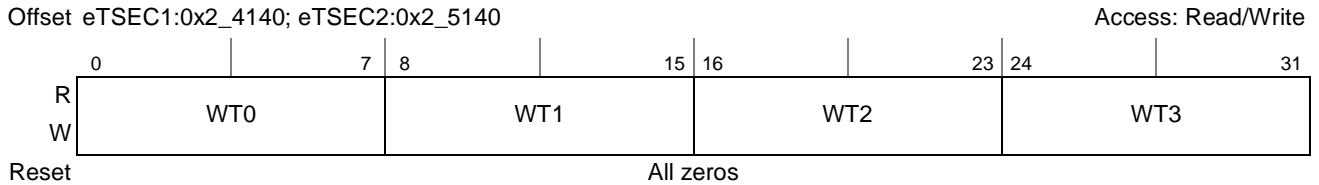
Table 16-20. TQUEUE Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16	EN0	Transmit queue 0 enable. 0 TxBD ring is not queried for transmission. In effect the transmit queue is disabled. 1 TxBD ring is queried for transmission.
17	EN1	Transmit queue 1 enable. 0 TxBD ring is not queried for transmission. In effect the transmit queue is disabled. 1 TxBD ring is queried for transmission.
18	EN2	Transmit queue 2 enable. 0 TxBD ring is not queried for transmission. In effect the transmit queue is disabled. 1 TxBD ring is queried for transmission.
19	EN3	Transmit queue 3 enable. 0 TxBD ring is not queried for transmission. In effect the transmit queue is disabled. 1 TxBD ring is queried for transmission.
20	EN4	Transmit queue 4 enable. 0 TxBD ring is not queried for transmission. In effect the transmit queue is disabled. 1 TxBD ring is queried for transmission.
21	EN5	Transmit queue 5 enable. 0 TxBD ring is not queried for transmission. In effect the transmit queue is disabled. 1 TxBD ring is queried for transmission.
22	EN6	Transmit queue 6 enable. 0 TxBD ring is not queried for transmission. In effect the transmit queue is disabled. 1 TxBD ring is queried for transmission.
23	EN7	Transmit queue 7 enable. 0 TxBD ring is not queried for transmission. In effect the transmit queue is disabled. 1 TxBD ring is queried for transmission.
24–31	—	Reserved

### 16.5.3.2.6 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 16.6.4.3.2, "Modified Weighted Round-Robin Queuing \(MWRR\)."](#) Figure 16-16 describes the TR03WT register.



**Figure 16-16. TR03WT Register Definition**

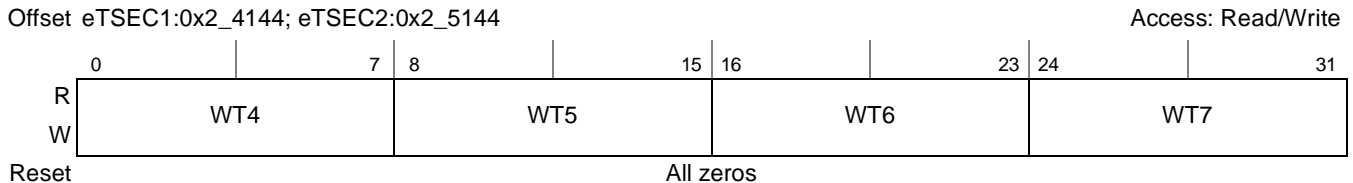
Table 16-21 describes the fields of the TR03WT register.

**Table 16-21. 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.

### 16.5.3.2.7 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 16.6.4.3.2, "Modified Weighted Round-Robin Queuing \(MWRR\)."](#) Figure 16-17 describes the definition for the TR47WT register.



**Figure 16-17. TR47WT Register Definition**

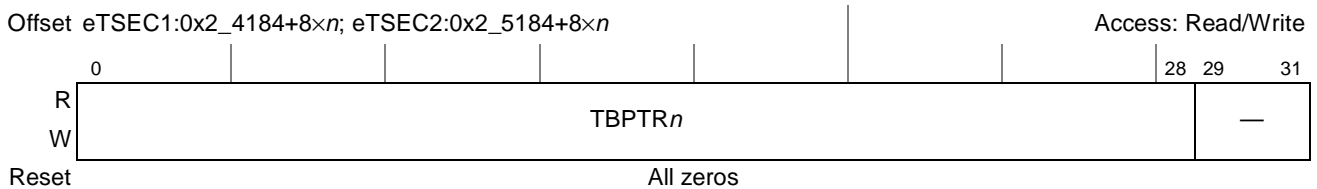
Table 16-22 describes the fields of the TR47WT register.

**Table 16-22. 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 $WT4 \times 64$ 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 $WT5 \times 64$ 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 $WT6 \times 64$ 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 $WT7 \times 64$ bytes of data are scheduled for transmission from TxBD ring 7. Clearing this field prevents transmission.

### 16.5.3.2.8 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 16-18 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 16-18. TBPTR0–TBPTR7 Register Definition**

Table 16-23 describes the fields of the TBPTR<sub>n</sub> register.

**Table 16-23. TBPTR<sub>n</sub> Field Descriptions**

Bits	Name	Description
0–28	TBPTR <sub>n</sub>	Current TxBD pointer for TxBD ring <i>n</i> . 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 <sub>n</sub> to the value in the corresponding TBASE <sub>n</sub> . 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 <sub>n</sub> to point to the first BD of the frame partially transmitted.
29–31	—	Reserved



### 16.5.3.2.9 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 16-19 describes the definition for the TBASE $n$  registers.

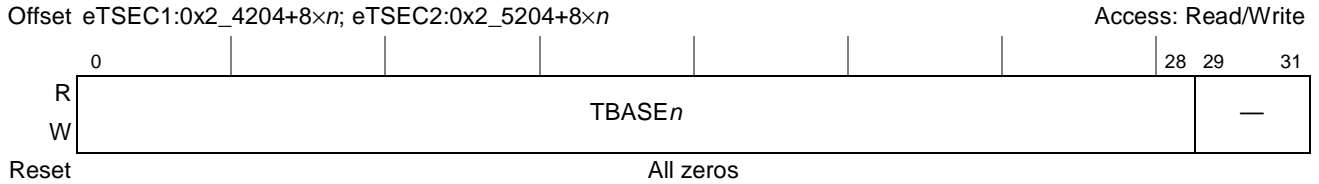


Figure 16-19. TBASE Register Definition

Table 16-24 describes the fields of the TBASE $n$  registers.

Table 16-24. 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

### 16.5.3.2.10 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 16-20 describes the definition for the TMR\_TXTS $n$ \_ID register.

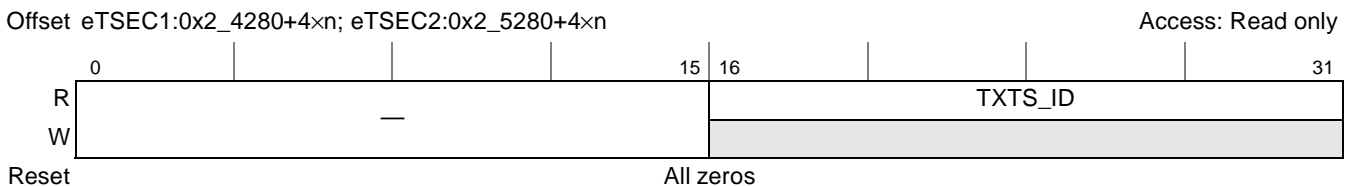


Figure 16-20. TMR\_TXTS $n$ \_ID Register Definition

Table 16-25 describes the fields of the TMR\_TXTS $n$ \_ID register.

Table 16-25. TMR\_TXTS $n$ \_ID Register Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–31	TXTS_ID	Tx time stamp identification field

### 16.5.3.2.11 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 16-21 depicts TMR\_TXTS $n$ \_H/L.

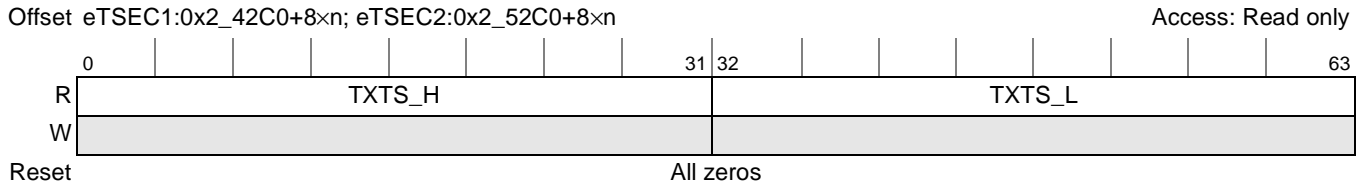


Figure 16-21. TMR\_TXTS $n$ \_H/L Register Definition

Table 16-26 describes the fields of the TMR\_TXTS $n$ \_H/L register.

Table 16-26. 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.

### 16.5.3.3 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.

#### 16.5.3.3.1 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 16-22 describes the RCTRL register.

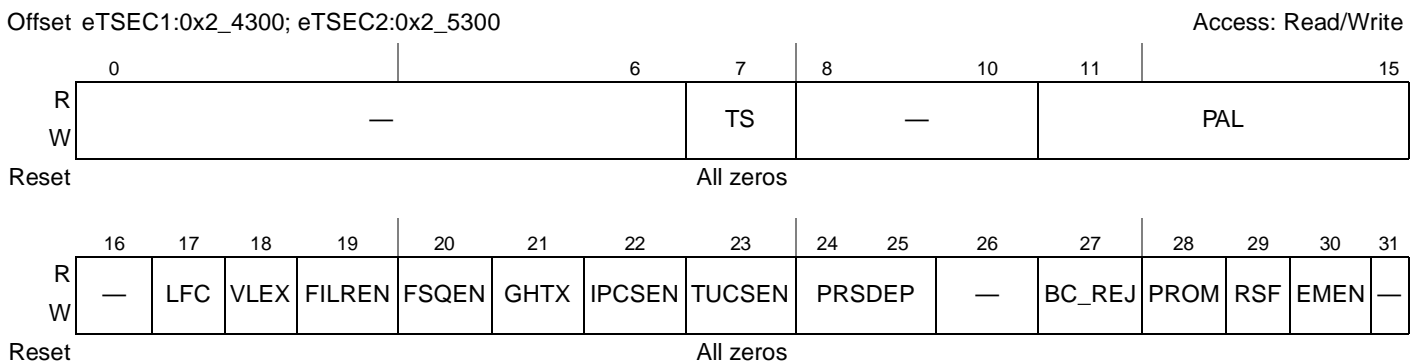


Figure 16-22. RCTRL Register Definition

Table 16-27 describes the fields of the RCTRL register.

**Table 16-27. RCTRL Field Descriptions**

Bits	Name	Description
0–6	—	Reserved
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. 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.
16	—	Reserved
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. 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. 0 Disabled. This is the default 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. 0 Do not delete VLAN tags from received Ethernet frames. 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. 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. 0 Do not search the receive queue filer table for received frames. All received frames are sent to RxBd ring 0 by default. 1 Search the receive queue filer table for received frames, and let the filer determine the index of the RxBd ring for each frame. 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. 0 The filer chooses the RxBd ring using the least significant bits of the virtual queue ID as a ring index. 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). 0 Both the individual and group hash functions are the 8 MSBs of the CRC-32 of the Ethernet destination address. 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.

Table 16-27. RCTRL Field Descriptions (continued)

Bits	Name	Description
22	IPCSSEN	IP Checksum verification enable. See <a href="#">Section 16.6.3.3, “Receive Path Off-Load.”</a> 0 IPv4 header checksums are not verified by the eTSEC—even if layer 3 parsing is enabled. 1 Perform IPv4 header checksum verification if PRSDEP > 01.
23	TUCSEN	TCP or UDP Checksum verification enable. See <a href="#">Section 16.6.3.3, “Receive Path Off-Load.”</a> 0 TCP or UDP checksums are not verified by the eTSEC—even if layer 4 parsing is enabled. 1 Perform TCP or UDP checksum verification if PRSDEP = 11.
24–25	PRSDEP	Parser control. The level of parser layer recognition is determined as follows: 00 Parser disabled. Receive frame filter must also be disabled by clearing RCTRL[FILREN]. 01 Only L2 (Ethernet) protocols are recognized. 10 L2 and L3 (IP) protocols are recognized. 11 L2, L3, and L4 (TCP/UDP) protocols are recognized. 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. 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	—	Reserved
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. 0 Ethernet frames less than 64 bytes in length are silently dropped. 1 Frames more than 16 bytes and less than 64 bytes in length are accepted upon a DA match. Note that frames less than or equal to 16 bytes 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

### 16.5.3.3.2 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 16-23 describes the definition for the RSTAT register.

Offset eTSEC1:0x2\_4304; eTSEC2:0x2\_5304

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 16-23. RSTAT Register Definition**

Table 16-28 describes the fields of the RSTAT register.

**Table 16-28. 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. 0 This queue is enabled for reception. (That is, it is not halted) 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. 0 This queue is enabled for reception. (That is, it is not halted) 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. 0 This queue is enabled for reception. (That is, it is not halted) 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. 0 This queue is enabled for reception. (That is, it is not halted) 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. 0 This queue is enabled for reception. (That is, it is not halted) 1 All controller receive activity to this queue is halted.

Table 16-28. 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. 0 This queue is enabled for reception. (That is, it is not halted) 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. 0 This queue is enabled for reception. (That is, it is not halted) 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. 0 This queue is enabled for reception. (That is, it is not halted) 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.

### 16.5.3.3.3 Receive Interrupt Coalescing Register (RXIC)

The RXIC register enables and configures the operational parameters for interrupt coalescing associated with received frames. Figure 16-24 describes the RXIC register.

Offset eTSEC1:0x2\_4310; eTSEC2:0x2\_5310

Access: Read/Write

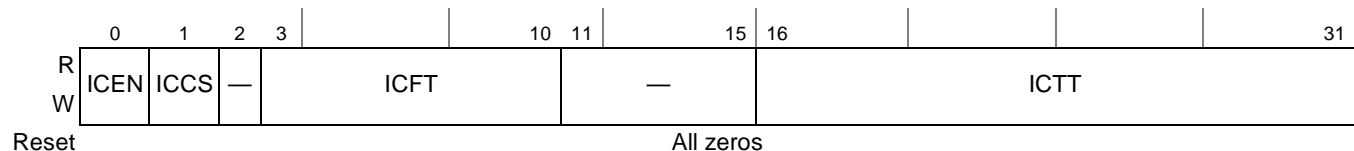


Figure 16-24. RXIC Register Definition

Table 16-29 describes the fields of the RXIC register.

**Table 16-29. RXIC Field Descriptions**

Bits	Name	Description
0	ICEN	Interrupt coalescing enable 0 Interrupt coalescing is disabled. Interrupts are raised as they are received. 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. 0 The coalescing timer advances count every 64 eTSEC Rx interface clocks (TSECn_GTX_CLK). 1 The coalescing timer advances count every 64 system clocks. 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 to 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.

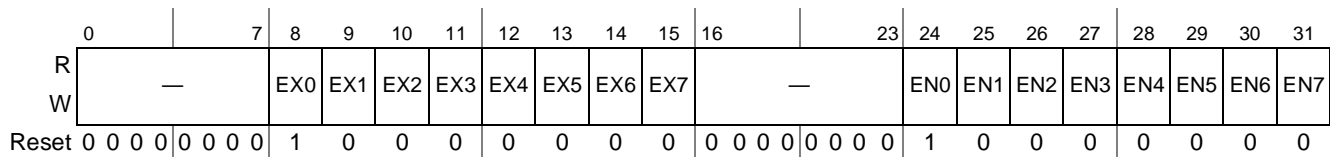
#### 16.5.3.3.4 Receive Queue Control Register (RQUEUE)

The RQUEUE register enables each of the RxBD rings 0–7. By default, RxBD ring 0 is enabled.

Figure 16-25 describes the definition for the RQUEUE register.

Offset eTSEC1:0x2\_4314; eTSEC2:0x2\_5314

Access: Read/Write



**Figure 16-25. RQUEUE Register Definition**

Table 16-30 describes the RQUEUE register.

**Table 16-30. RQUEUE Field Descriptions**

Bits	Name	Description
0–7	—	Reserved
8–15	EX $n$	Receive queue $n$ extract enable. 0 Data transferred by DMA to this RxBD ring is not extracted to cache. 1 Data transferred by DMA to this RxBD ring undergoes extraction according to ATTR register.

Table 16-30. RQUEUE Field Descriptions (continued)

Bits	Name	Description
16–23	—	Reserved
24–31	EN $n$	Receive queue $n$ enable. 0 RxB $D$ ring is not queried for reception. In effect the receive queue is disabled. 1 RxB $D$ ring is queried for reception.

### 16.5.3.3.5 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 bit fields 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 16-26 describes the definition for the RBIFX register.

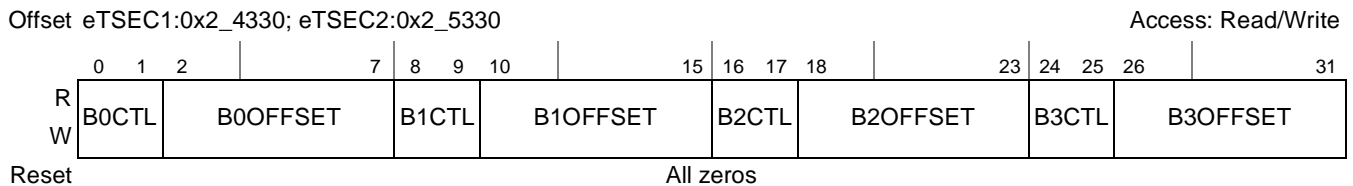


Figure 16-26. RBIFX Register Definition

Table 16-31 describes the RBIFX register.

Table 16-31. RBIFX Field Descriptions

Bits	Name	Description
0–1	B0CTL	Location of byte 0 of property ARB. 00 Byte 0 is not extracted, and appears as zero in property ARB. 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. 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. 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.

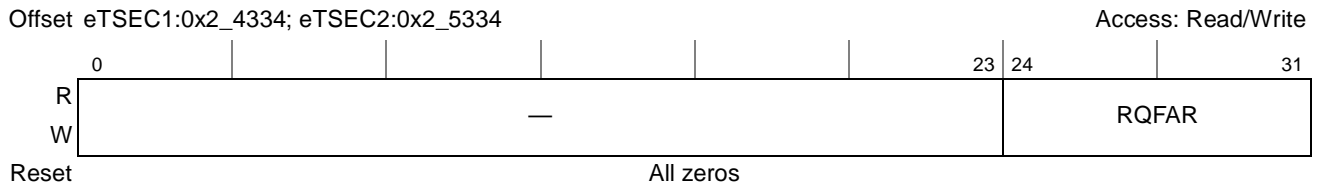


Table 16-31. RBIFX Field Descriptions (continued)

Bits	Name	Description
8–9	B1CTL	Location of byte 1 of property ARB. 00 Byte 1 is not extracted, and appears as zero in property ARB. 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. 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. 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. 00 Byte 2 is not extracted, and appears as zero in property ARB. 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. 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. 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.
24–25	B3CTL	Location of byte 3 of property ARB. 00 Byte 3 is not extracted, and appears as zero in property ARB. 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. 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. 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.

### 16.5.3.3.6 Receive Queue Filer Table Address Register (RQFAR)

RQFAR, shown in Figure 16-27, 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.



**Figure 16-27. Receive Queue Filer Table Address Register Definition**

Table 16-32 describes the fields of the RQFAR register.

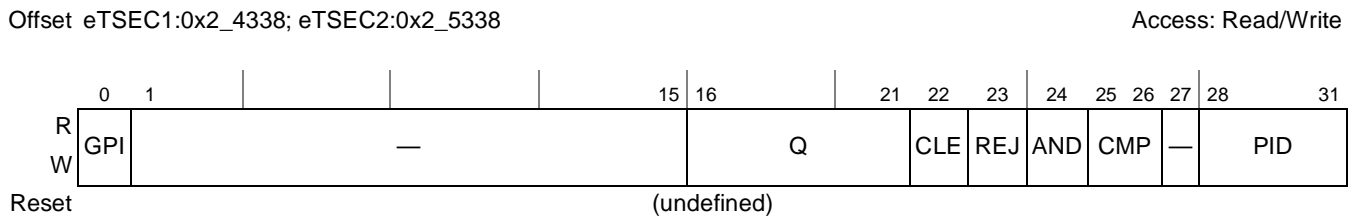
**Table 16-32. RQFAR Field Descriptions**

Bits	Name	Description
0–23	—	Reserved
24–31	RQFAR	Current index of receive queue filer table, which spans a total of 256 entries.

### 16.5.3.3.7 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 16.6.4.2, “Receive Queue Filer.”](#) The word accessed through RQFCR is defined by the current value of RQFAR.

Figure 16-28 describes the definition for the RQFCR register.



**Figure 16-28. Receive Queue Filer Table Control Register Definition**

Table 16-33 describes the fields of the RQFCR register.

**Table 16-33. 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. 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.

Table 16-33. RQFCR Field Descriptions (continued)

Bit	Name	Description
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. 0 Regular RQCTRL entry. 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. 0 If entry matches, accept frame and file it to RxBD ring Q. 1 If entry matches, reject frame and discard it, ignoring Q.
24	AND	Match this entry and the next entry as a pair. 0 Match property[PID] against RQPROP, independent of the next entry. 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).
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 <i>mask_register</i> prior to comparison against RQPROP. However, the property value is not permanently altered by the value in <i>mask_register</i> . By default, <i>mask_register</i> is initialized to 0xFFFF_FFFF before each frame is processed.  In the case where PID = 0, CMP is interpreted as follows: 00/01 Filter <i>mask_register</i> is set to all 32 bits of RQPROP, and this entry always <i>matches</i> . 10/11 Filter <i>mask_register</i> is set to all 32 bits of RQPROP, and this entry always <i>fails to match</i> .  In the case where PID > 0, CMP is interpreted as follows (& is bit-wise AND operator): 00 <i>property</i> [PID] & <i>mask_register</i> = RQPROP 01 <i>property</i> [PID] & <i>mask_register</i> ≥ RQPROP 10 <i>property</i> [PID] & <i>mask_register</i> ≠ RQPROP 11 <i>property</i> [PID] & <i>mask_register</i> < 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 16-34).

### 16.5.3.3.8 Receive Queue Filer Table Property Register (RQFPR)

RQFPR (see Figure 16-29) 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 16.6.4.2, “Receive Queue Filer.” The word accessed through RQFPR is defined by the current value of RQFAR. Figure 16-29 and Figure 16-30 describe the fields of the RQFPR register according to property ID.

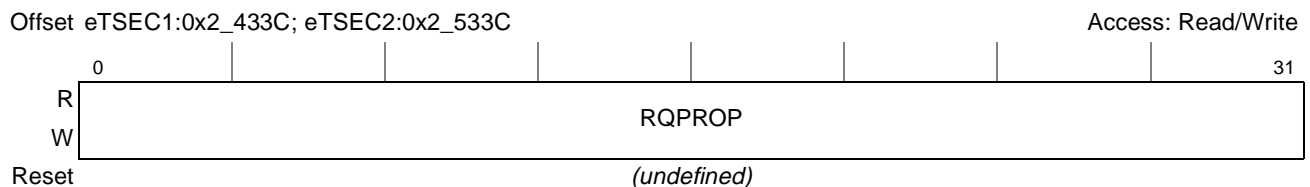


Figure 16-29. Receive Queue Filer Table Property IDs 0, 2–15 Register Definition

Offset eTSEC1:0x2\_433C; eTSEC2:0x2\_533C

Access: Read/Write

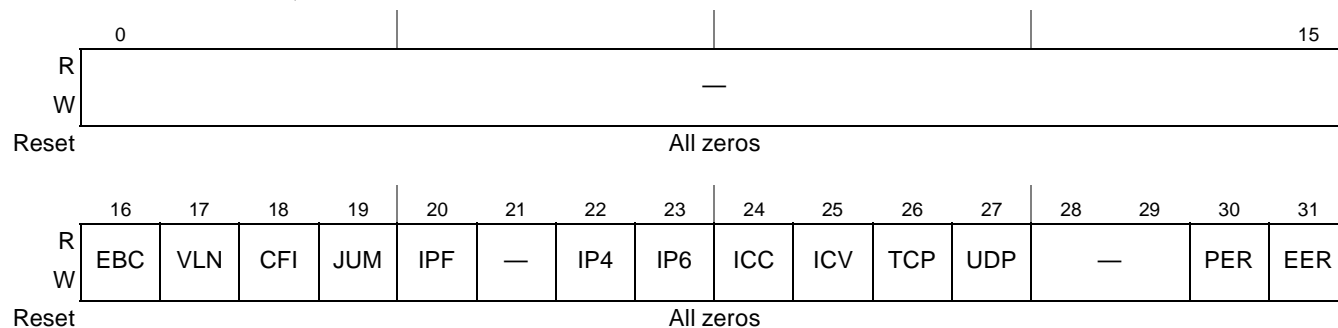


Figure 16-30. Receive Queue Filer Table Property ID1 Register Definition

Table 16-34 describes the fields of the RQFPR register.

Table 16-34. RQFPR Field Descriptions

PID <sup>1</sup>	Bit	Name	Description
0000	0–31	MASK	Mask bits to be written to Filer <i>mask_register</i> for masking of property values. The rule match/fail status for this PID is determined by RQCTRL[ <i>CMP</i> ]. Since <i>mask_register</i> 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	0–15	—	Reserved
	16	EBC	Set if the destination Ethernet address is to the broadcast address.
	17	VLN	Set if a VLAN tag (Ethertype DFVLAN[ <i>TAG</i> ] 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. See the descriptions of receive FCB fields IP and PRO in <a href="#">Section 16.6.3.3, “Receive Path Off-Load,”</a> for more information on determining the status of received packets for which IPF is set.
	21	—	Reserved
	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.	

Table 16-34. RQFPR Field Descriptions (continued)

PID <sup>1</sup>	Bit	Name	Description
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.
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.

Table 16-34. RQFPR Field Descriptions (continued)

PID <sup>1</sup>	Bit	Name	Description
0111	0–15	—	Reserved, should be written with zero.
	16–31	ETY	<p>Ethertype of next layer protocol, that is, last ethertype if layer 2 headers nest. Defaults to 0xFFFF.</p> <p>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.</p> <p>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.</p> <p>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.</p> <p>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.</p> <ul style="list-style-type: none"> <li>• 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. In this case, one can use arbitrary extracted bytes to pull the outermost Ethertype.</li> <li>• The PPPoE ethertype described above.</li> <li>• 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.</li> <li>• 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. NOTE</li> </ul> <p>Users of the eTSEC parser/filer should be aware of a difference in behavior between rev 1 and rev 2 silicon in cases where the Ethernet type/length field contains a value between 1500 and 1536. In rev 2 silicon, values between 1500 and 1536 are interpreted as a type. Since there are currently no valid types in this range publicly defined by IANA, the controller will not parse beyond the length/type field of any such frame.</p> <p>If the same packet is encountered with rev 1 silicon, parser/filer behavior is different. With rev 1 silicon, such packets are treated as payload length. S/W 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.</p>
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	<p>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.</p> <p>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.</p>

Table 16-34. RQFPR Field Descriptions (continued)

PID <sup>1</sup>	Bit	Name	Description
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).
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.

<sup>1</sup> PID is the property identifier field of the filer table control entry (see RQFCR[PID]) at the same index.

### 16.5.3.3.9 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 16-31](#) describes the definition for the MRBLR.

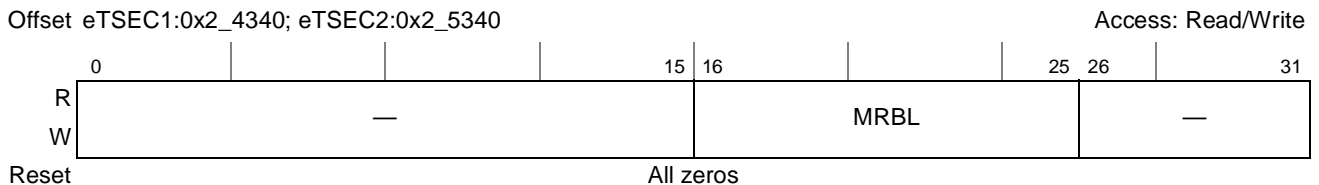


Figure 16-31. MRBLR Register Definition

Table 16-35. 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 <a href="#">Section 16.5.3.5.5, “Maximum Frame Length Register (MAXFRM),”</a> for further discussion.
26–31	—	To ensure that MRBL is a multiple of 64, these bits are reserved and should be cleared.

### 16.5.3.3.10 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 16-32 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.

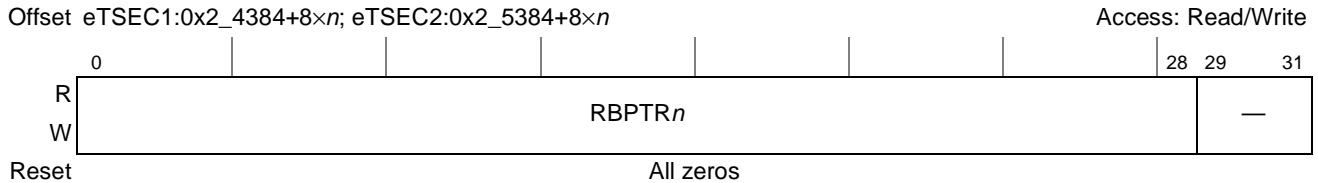


Figure 16-32. RBPTR0–RBPTR7 Register Definition

Table 16-36 describes the fields of the RBPTR $n$  register.

Table 16-36. 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

### 16.5.3.3.11 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 16-33 describes the RBASE $n$  registers.

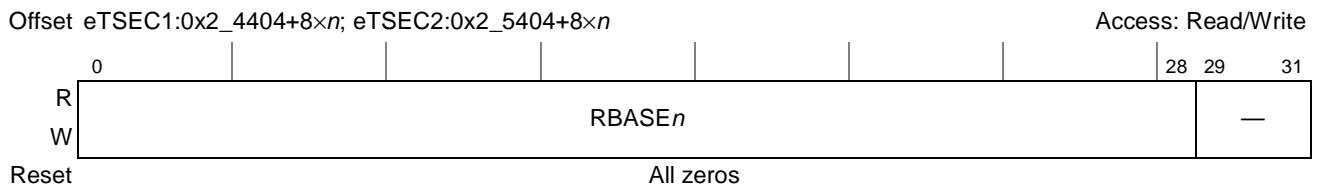


Figure 16-33. RBASE Register Definition



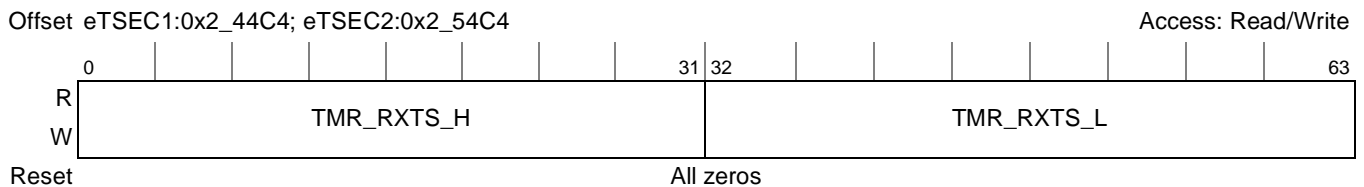
Table 16-37 describes the fields of the RBASE $n$  registers.

**Table 16-37. 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

### 16.5.3.3.12 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 16-34 describes the definition for the RXTS\_H/L register.



**Figure 16-34. TMR\_RXTS\_H/L Register Definition**

Table 16-38 describes the fields of the TMR\_RXTS\_H/L register.

**Table 16-38. 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.

## 16.5.3.4 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.

### 16.5.3.4.1 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 be 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.

#### 16.5.3.4.2 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.

#### 16.5.3.4.3 Handling Packet Collisions

While transmitting a packet in half-duplex mode, the eTSEC is sensitive to COL. 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:

$$0 \leq r \leq 2^k, \text{ where } k = \min(n, 10).$$

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.

#### 16.5.3.4.4 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.

#### 16.5.3.4.5 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.

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.

### 16.5.3.5 MAC Registers

This section describes the MAC registers.

#### 16.5.3.5.1 MAC Configuration 1 Register (MACCFG1)

MACCFG1 is written by the user. Figure 16-35 describes the definition for the MACCFG1 register.

Offset eTSEC1:0x2\_4500; eTSEC2:0x2\_5500

Access: Mixed

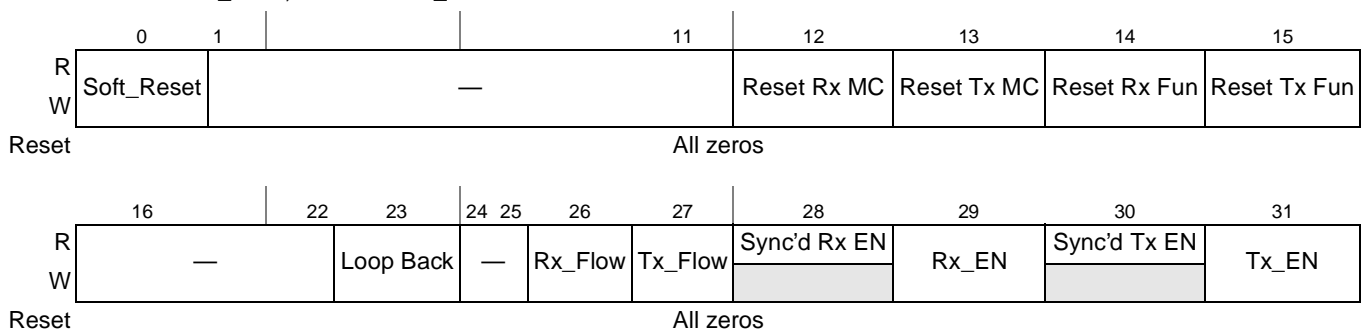


Figure 16-35. MACCFG1 Register Definition

Table 16-39 describes the fields of the MACCFG1 register.

Table 16-39. MACCFG1 Field Descriptions

Bits	Name	Description
0	Soft_Reset	Soft reset. This bit is cleared by default. See <a href="#">Section 16.6.2.2, “Soft Reset and Reconfiguring Procedure,”</a> for more information on setting this bit. 0 Normal operation. 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. 0 Normal operation. 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. 0 Normal operation. 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. 0 Normal operation. 1 Place the receive function in reset. This block performs the receive frame protocol.

Table 16-39. MACCFG1 Field Descriptions (continued)

Bits	Name	Description
15	Reset Tx Fun	Reset transmit function block. This bit is cleared by default. 0 Normal operation. 1 Place the transmit function in reset. This block performs the frame transmission protocol.
16–22	—	Reserved
23	Loop Back	Loop back. This bit is cleared by default. 0 Normal operation. 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. Must be 0 if MACCFG2[Full Duplex] = 0. 0 The receive MAC control ignores PAUSE flow control frames. 1 The receive MAC control detects and acts on PAUSE flow control frames. <b>Note:</b> Should not be set when operating in Half-Duplex mode
27	Tx_Flow	Transmit flow. This bit is cleared by default. Must be 0 if MACCFG2[Full Duplex] = 0. 0 The transmit MAC control may not send PAUSE flow control frames if requested by the system. <b>Note:</b> 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) 0 Frame reception is not enabled. 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). 0 The MAC may not receive frames from the PHY. 1 The MAC may receive frames from the PHY.
30	Sync'd Tx EN	Transmit enable synchronized to the transmit stream. (Read-only) 0 Frame transmission is not enabled. 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). 0 The MAC may not transmit frames from the system. 1 The MAC may transmit frames from the system.

### 16.5.3.5.2 MAC Configuration 2 Register (MACCFG2)

The MACCFG2 register is written by the user. Figure 16-36 describes the definition for the MACCFG2 register.

Offset eTSEC1:0x2\_4504; eTSEC2:0x2\_5504

Access: Read/Write

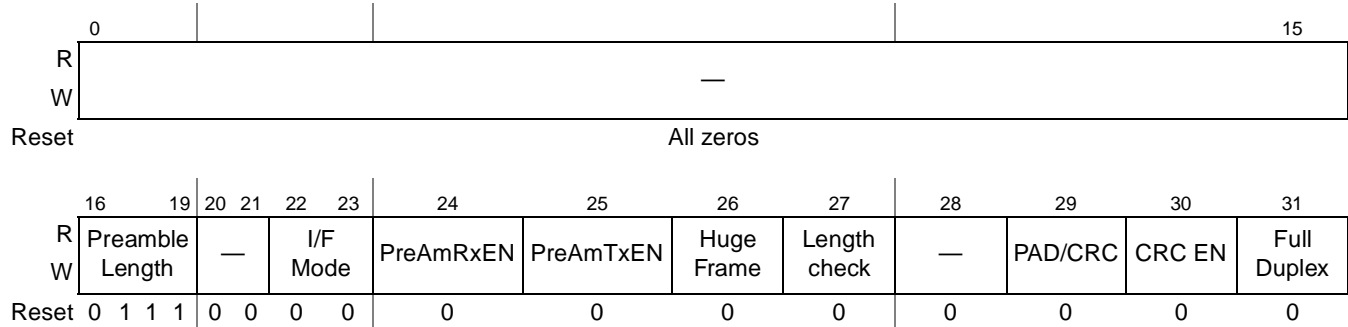


Figure 16-36. MACCFG2 Register Definition

Table 16-40 describes the fields of the MACCFG2 register.

Table 16-40. 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. 00 Reserved bit mode (not supported) (10 Mbps GENDEC/GPSI) 01 Nibble mode (MII, RGMII) (10/100 Mbps MII, RGMII). For RGMII, ECNTRL.RPM should be set to 1. 10 Reserved 11 Reserved
24	PreAM RxEN	User defined preamble enable for received frames. This bit is cleared by default. 0 The MAC skips the Ethernet preamble without returning it. 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, 0s are prepended to pad it to 7 bytes.
25	PreAM TxEN	User defined preamble enable for transmitted frames. This bit is cleared by default. 0 The MAC generates a standard Ethernet preamble. 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.

Table 16-40. MACCFG2 Field Descriptions (continued)

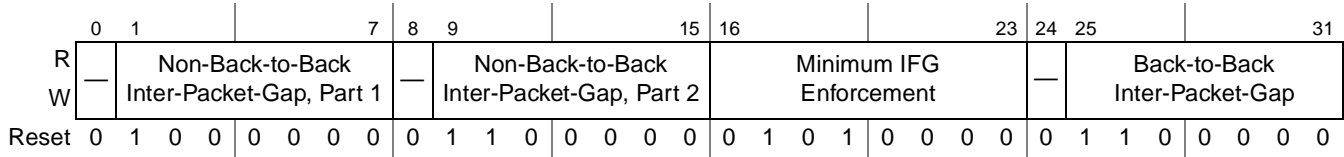
Bits	Name	Description																				
26	Huge Frame	<p>Huge frame enable. This bit is cleared by default.</p> <p>0 Limit the length of frames received to less than or equal to the maximum frame length value (MAXFRM[Maximum Frame]) and limit the length of frames transmitted to less than the maximum frame length. See <a href="#">Section 16.6.7, “Buffer Descriptors,”</a> for further details of buffer descriptor bit updating.</p> <table border="1"> <thead> <tr> <th>Frame type</th> <th>Frame length</th> <th>Packet truncation</th> <th>Buffer descriptor updated</th> </tr> </thead> <tbody> <tr> <td>Receive or transmit</td> <td>&gt; 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>&lt; maximum frame length</td> <td>no</td> <td>no</td> </tr> </tbody> </table> <p>1 Frames are transmitted and received regardless of their relationship to the maximum frame length. Note that if Huge Frame is cleared, the user must ensure that adequate buffer space is allocated for received frames. See <a href="#">Section 16.5.3.5.5, “Maximum Frame Length Register (MAXFRM),”</a> for further information.</p>	Frame type	Frame length	Packet truncation	Buffer descriptor updated	Receive or transmit	> maximum frame length	yes	yes	Receive	= maximum frame length	no	no	Transmit	= maximum frame length	no	yes	Receive or transmit	< maximum frame length	no	no
Frame type	Frame length	Packet truncation	Buffer descriptor updated																			
Receive or transmit	> maximum frame length	yes	yes																			
Receive	= maximum frame length	no	no																			
Transmit	= maximum frame length	no	yes																			
Receive or transmit	< maximum frame length	no	no																			
27	Length check	<p>Length check. This bit is cleared by default.</p> <p>0 No length field checking is performed.</p> <p>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.</p>																				
28	—	Reserved																				
29	PAD/CRC	<p>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).</p> <p>0 Frames presented to the MAC have a valid length and contain a CRC.</p> <p>1 The MAC pads all transmitted short frames and appends a CRC to every frame regardless of padding requirement.</p>																				
30	CRC EN	<p>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.</p> <p>0 Frames presented to the MAC have a valid length and contain a valid CRC.</p> <p>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.</p>																				
31	Full Duplex	<p>Full duplex configure. This bit is cleared by default.</p> <p>0 The MAC operates in half-duplex mode only.</p> <p>1 The MAC operates in full-duplex mode.</p>																				

### 16.5.3.5.3 Inter-Packet Gap/Inter-Frame Gap Register (IPGIFG)

The IPGIFG register is written by the user. [Figure 16-37](#) describes the definition for IPGIFG.

Offset eTSEC1:0x2\_4508; eTSEC2:0x2\_5508

Access: Read/Write



**Figure 16-37. IPGIFG Register Definition**

[Table 16-41](#) describes the fields of the IPGIFG register.

**Table 16-41. 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.



### 16.5.3.5.4 Half-Duplex Register (HAFDUP)

The HAFDUP register is written by the user. Figure 16-38 describes the HAFDUP register.

Offset eTSEC1:0x2\_450C; eTSEC2:0x2\_550C

Access: Read/Write

	0	7	8	11	12	13	14	15	
R	—			Alternate BEB Truncation	Alt BEB	BP No BackOff	No BackOff	Excess Defer	
W	—			Alternate BEB Truncation	Alt BEB	BP No BackOff	No BackOff	Excess Defer	
Reset	0	0	0	0	1	0	1	0	
	16	19	20	21	22	31			
R	Retransmission Maximum			—			Collision Window		
W	Retransmission Maximum			—			Collision Window		
Reset	1	1	1	1	0	0	0	1	

Figure 16-38. Half-Duplex Register Definition

Table 16-42 describes the fields of the HAFDUP register.

Table 16-42. 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. 0 The Tx MAC follows the standard binary exponential back off rule. 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. 0 The Tx MAC follows the binary exponential back off rule. 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. 0 The Tx MAC follows the binary exponential back off rule. 1 The Tx MAC immediately re-transmits following a collision.
15	Excess Defer	Excessively deferred. This bit is set by default. 0 The Tx MAC aborts the transmission of a packet that is excessively deferred. 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.
20–21	—	Reserved
22–31	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.





### 16.5.3.5.8 MII Management Address Register (MIIMADD)

The MIIMADD register is written by the user. Figure 16-42 shows the MIIMADD register.

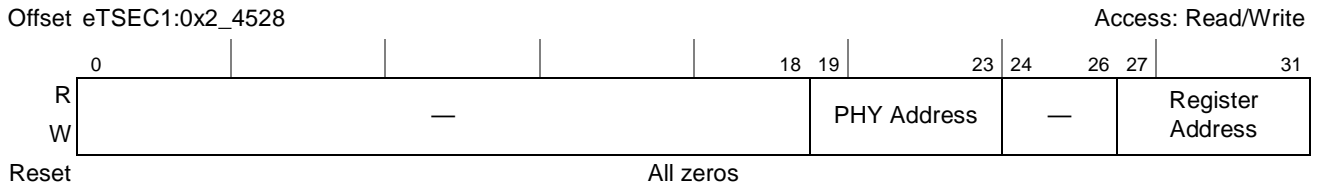


Figure 16-42. MIIMADD Register Definition

Table 16-46 describes the fields of the MIIMADD register.

Table 16-46. 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.

### 16.5.3.5.9 MII Management Control Register (MIIMCON)

MIIMCON, shown in Figure 16-43, is written by the user.

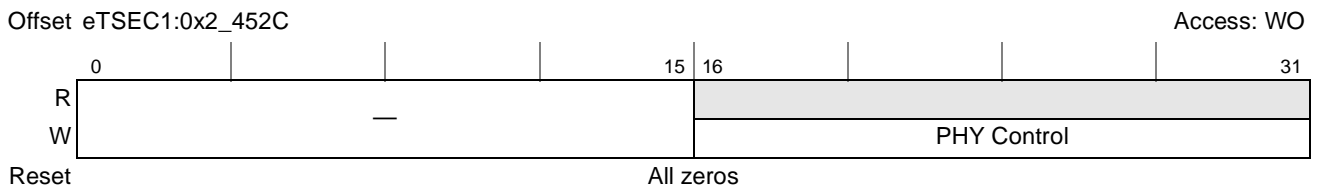


Figure 16-43. MII Mgmt Control Register Definition

Table 16-47 describes the fields of the MIIMCON register.

Table 16-47. 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.

### 16.5.3.5.10 MII Management Status Register (MIIMSTAT)

The MIIMSTAT register is read only by the user. Figure 16-44 describes the definition for the MIIMSTAT register.

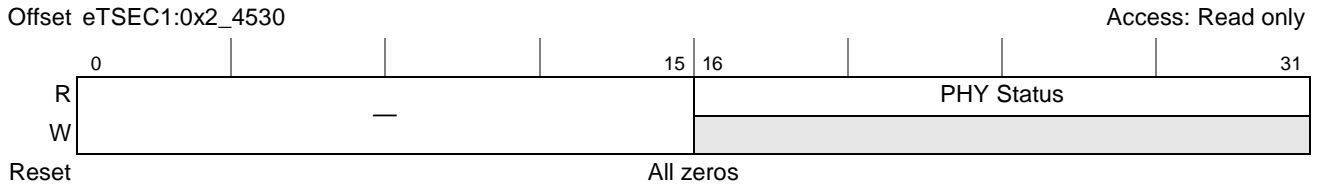


Figure 16-44. MIIMSTAT Register Definition

Table 16-48 describes the fields of the MIIMSTAT register.

Table 16-48. 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.

### 16.5.3.5.11 MII Management Indicator Register (MIIMIND)

The MIIMIND register is read-only by the user. Figure 16-45 describes the definition for the MIIMIND register.

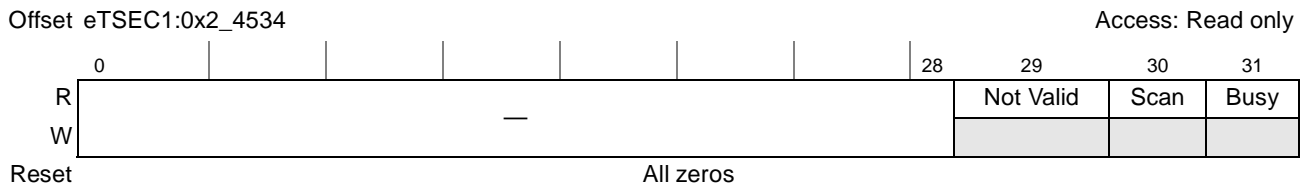


Figure 16-45. MII Mgmt Indicator Register Definition

Table 16-49. MIIMIND Field Descriptions

Bits	Name	Description
0–28	—	Reserved
29	Not Valid	Not valid. 0 MII Mgmt read cycle has completed and the read data is valid. 1 MII Mgmt read cycle has not completed and the read data is not yet valid.
30	Scan	Scan in progress. 0 A scan operation (continuous MII Mgmt read cycles) is not in progress. 1 A scan operation (continuous MII Mgmt read cycles) is in progress.
31	Busy	Busy. 0 MII Mgmt block is not currently performing an MII Mgmt read or write cycle. 1 MII Mgmt block is currently performing an MII Mgmt read or write cycle.

### 16.5.3.5.12 Interface Status Register (IFSTAT)

Figure 16-46 shows the IFSTAT register.

Offset eTSEC1:0x2\_453C; eTSEC2:0x2\_553C

Access: Read only

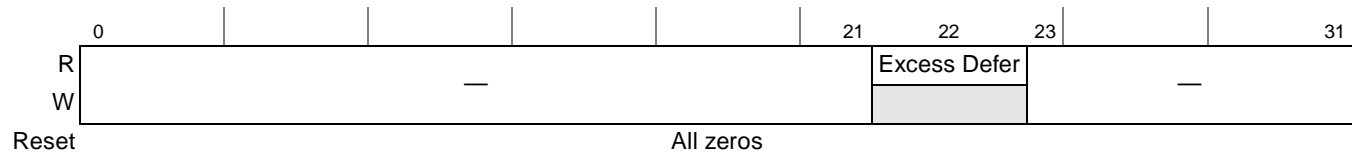


Figure 16-46. Interface Status Register Definition

Table 16-50 describes the fields of the FSTAT register.

Table 16-50. 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. 0 Normal operation. 1 The MAC excessively defers a transmission.
23–31	—	Reserved

### 16.5.3.5.13 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 0x1234\_5678\_ABCD, MACSTNADDR1 is set to 0xCDAB\_7856 and MACSTNADDR2 is set to 0x3412\_0000.

Figure 16-47 shows the MACSTNADDR1 register.

Offset eTSEC1:0x2\_4540; eTSEC2:0x2\_5540

Access: Read/Write

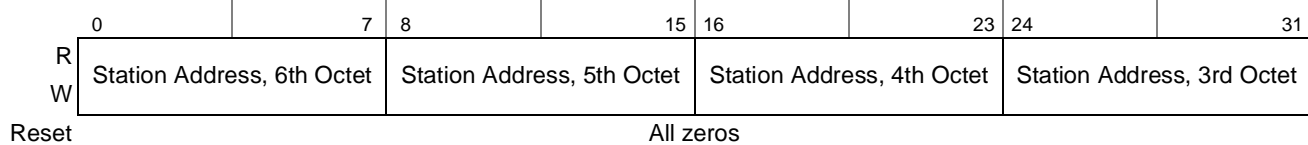


Figure 16-47. MAC Station Address Part 1 Register Definition

Table 16-51 describes the fields of the MACSTNADDR1 register.

Table 16-51. MACSTNADDR1 Field Descriptions

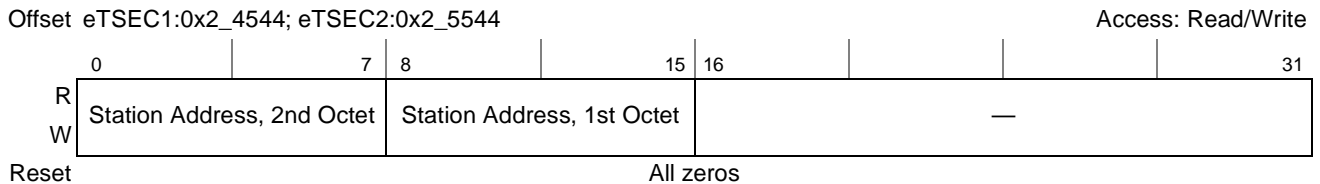
Bit	Name	Description
0–7	Station Address, 6th Octet	This field holds the sixth octet of the station address. The sixth octet (station address bits 40–47) defaults to a value of 0x0.
8–15	Station Address, 5th Octet	This field holds the fifth octet of the station address. The fifth octet (station address bits 32–39) defaults to a value of 0x0.

**Table 16-51. MACSTNADDR1 Field Descriptions (continued)**

Bit	Name	Description
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.

#### 16.5.3.5.14 MAC Station Address Part 2 Register (MACSTNADDR2)

The MACSTNADDR2 register is written by the user. [Figure 16-48](#) describes the definition for the MACSTNADDR2 register.

**Figure 16-48. MAC Station Address Part 2 Register Definition**

[Table 16-52](#) describes the fields of the MACSTNADDR2 register.

**Table 16-52. 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

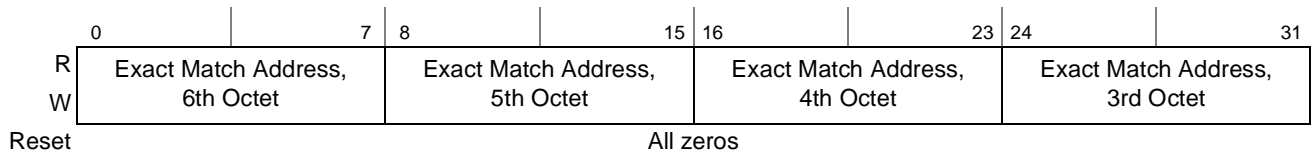
#### 16.5.3.5.15 MAC Exact Match Address 1–15 Part 1 Registers (MAC01ADDR1–MAC15ADDR1)

The MAC01ADDR1–MAC15ADDR1 registers are written by the user with the unicast or multicast addresses aliasing the MAC. [Figure 16-49](#) describes the definition for all of the fifteen MAC<sub>n</sub>ADDR1 registers. The value of the address written into MAC<sub>x</sub>ADDR1 and MAC<sub>n</sub>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 0x1234\_5678\_ABCD, MAC<sub>n</sub>ADDR1 is set to 0xCDAB\_7856 and MAC<sub>n</sub>ADDR2 is set to 0x3412\_0000.

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*; eTSEC2:0x2\_5548+8×*n*

Access: Read/Write



**Figure 16-49. MAC Exact Match Address *n* Part 1 Register Definition**

Table 16-53 describes the fields of a MAC<sub>*n*</sub>ADDR1 register.

**Table 16-53. MAC<sub>*n*</sub>ADDR1 Field Descriptions**

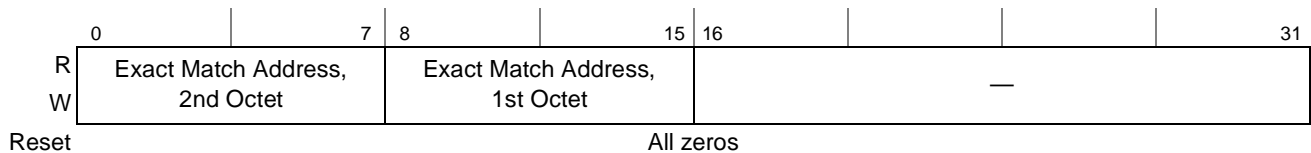
Bit	Name	Description
0–7	Exact Match Address, 6th Octet	Holds the sixth octet of the exact match address. The sixth octet (destination address bits 40–47) defaults to a value of 0x0.
8–15	Exact Match Address, 5th Octet	Holds the fifth octet of the exact match address. The fifth octet (destination address 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.

### 16.5.3.5.16 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 16-50 describes the definition for all of the fifteen MAC<sub>*x*</sub>ADDR2 registers.

Offset eTSEC1:0x2\_454C+8×*n*; eTSEC2:0x2\_554C+8×*n*

Access: Read/Write



**Figure 16-50. MAC Exact Match Address *x* Part 2 Register Definition**



Table 16-54 describes the fields of a MAC<sub>x</sub>ADDR2 register.

**Table 16-54. 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

### 16.5.3.6 MIB Registers

This section describes the MIB registers. The eTSEC RMON module has 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.

#### NOTE

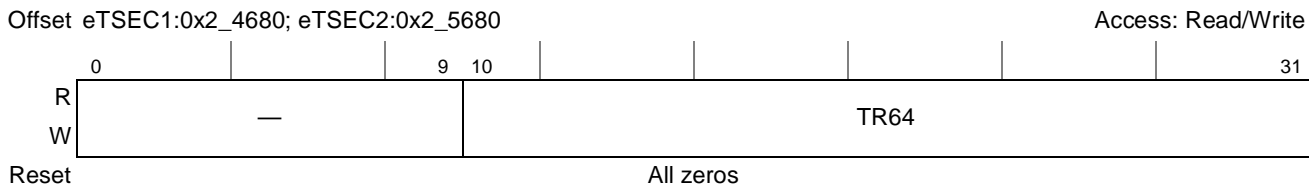
RMON counters do not comprehend custom VLAN tagged frames. Affected counters include TRMGV, RMCA, RBCA, RXCF, RXPf, RXUO, RALN, RFLR, ROVR, RJBR, TMCA, TBCA, TXPF, TXCF. Specifically, custom VLAN tagged frames are not afforded the ability to be greater than 1518, as compared to the IEEE standard tagged frames.

#### NOTE

The transmit and receive frame counters (TR64, TR127, TR 255, TR511, TR1K, TRMAX, and TRMGV) do not increment for aborted frames (collision retry limit exceeded, late collision, underrun, EBERR, Tx FIFO data error, frame truncated due to exceeding MAXFRM, or excessive deferral).

### 16.5.3.6.1 Transmit and Receive 64-Byte Frame Counter (TR64)

Figure 16-51 describes the definition for the TR64 register.



**Figure 16-51. Transmit and Receive 64-Byte Frame Register Definition**

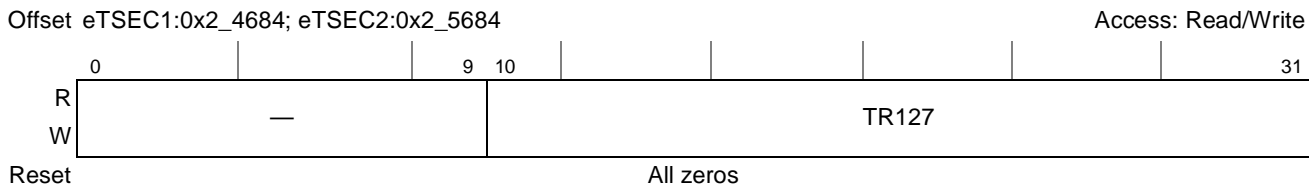
Table 16-55 describes the fields of the TR64 register.

**Table 16-55. 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).

### 16.5.3.6.2 Transmit and Receive 65- to 127-Byte Frame Counter (TR127)

Figure 16-52 describes the definition for the TR127 register.



**Figure 16-52. Transmit and Receive 65- to 127-Byte Frame Register Definition**

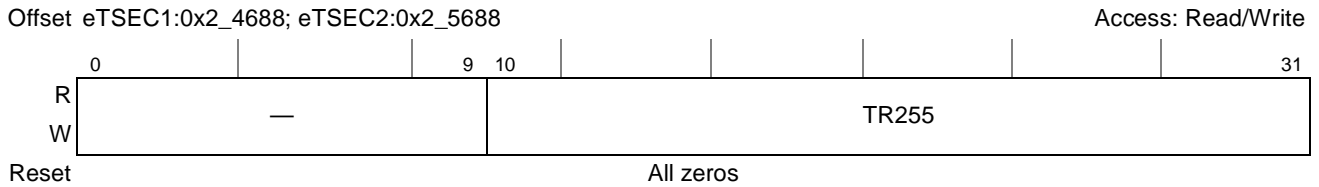
Table 16-56 describes the fields of the TR127 register.

**Table 16-56. 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).

### 16.5.3.6.3 Transmit and Receive 128- to 255-Byte Frame Counter (TR255)

Figure 16-53 describes the definition for the TR255 register.



**Figure 16-53. Transmit and Received 128- to 255-Byte Frame Register Definition**

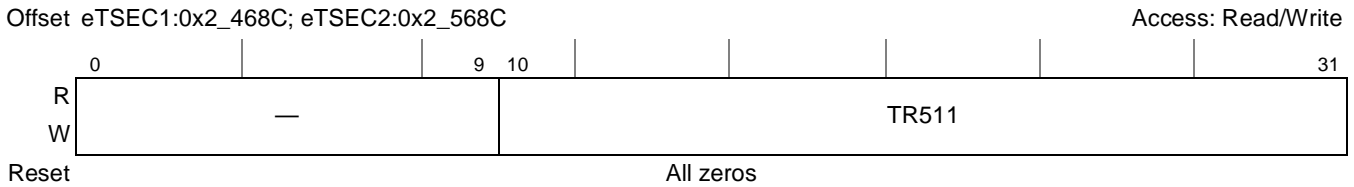
Table 16-57 describes the fields of the TR255 register.

**Table 16-57. 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).

### 16.5.3.6.4 Transmit and Receive 256- to 511-Byte Frame Counter (TR511)

Figure 16-54 describes the definition for the TR511 register.



**Figure 16-54. Transmit and Received 256- to 511-Byte Frame Register Definition**

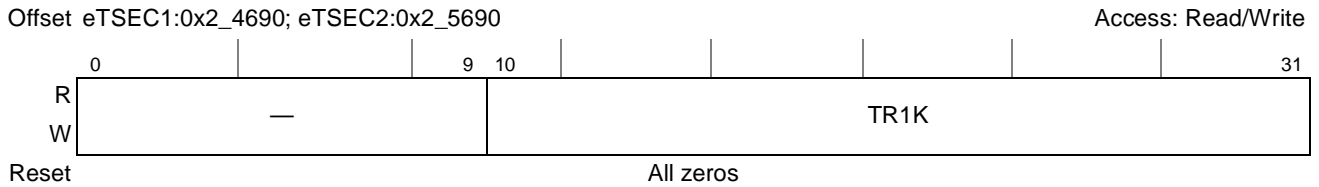
Table 16-58 describes the fields of the TR511 register.

**Table 16-58. 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).

### 16.5.3.6.5 Transmit and Receive 512- to 1023-Byte Frame Counter (TR1K)

Figure 16-55 shows the TR1K register.



**Figure 16-55. Transmit and Received 512- to 1023-Byte Frame Register Definition**

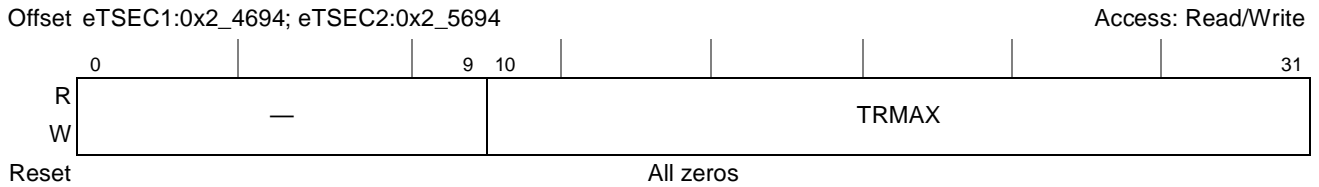
Table 16-59 describes the fields of the TR1K register.

**Table 16-59. 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).

### 16.5.3.6.6 Transmit and Receive 1024- to 1518-Byte Frame Counter (TRMAX)

Figure 16-56 describes the definition for the TRMAX register.



**Figure 16-56. Transmit and Received 1024- to 1518-Byte Frame Register Definition**

Table 16-60 describes the fields of the TRMAX register.

**Table 16-60. 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).

### 16.5.3.6.7 Transmit and Receive 1519- to 1522-Byte VLAN Frame Counter (TRMGV)

Figure 16-57 describes the definition for the TRMGV register.

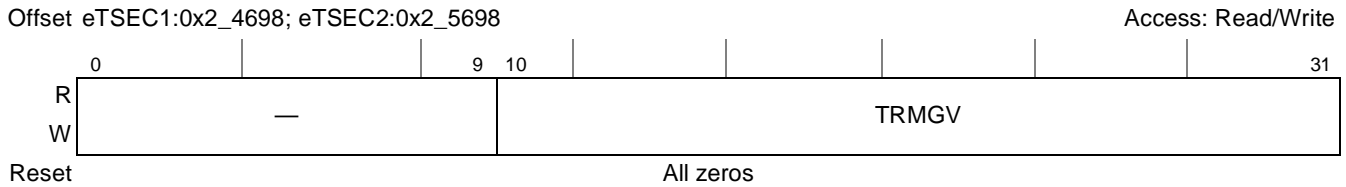


Figure 16-57. Transmit and Received 1519- to 1522-Byte VLAN Frame Register Definition

Table 16-61 describes the fields of the TRMGV register.

Table 16-61. 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).

### 16.5.3.6.8 Receive Byte Counter (RBYT)

Figure 16-58 shows the RBYT register.

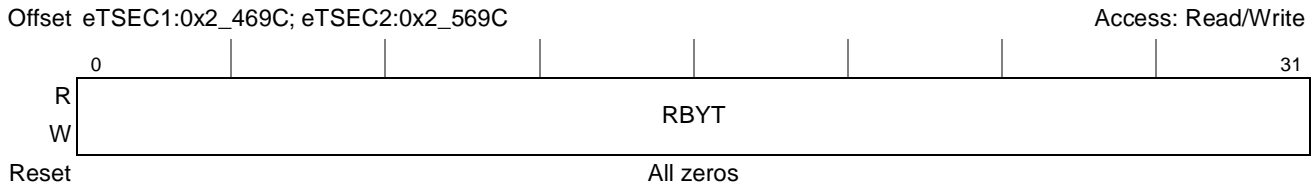


Figure 16-58. Receive Byte Counter Register Definition

Table 16-62 describes the fields of the RBYT register.

Table 16-62. 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.

### 16.5.3.6.9 Receive Packet Counter (RPKT)

Figure 16-59 describes the definition for the RPKT register.

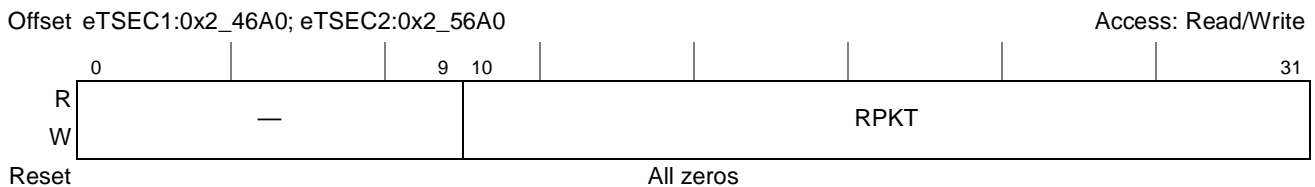


Figure 16-59. Receive Packet Counter Register Definition

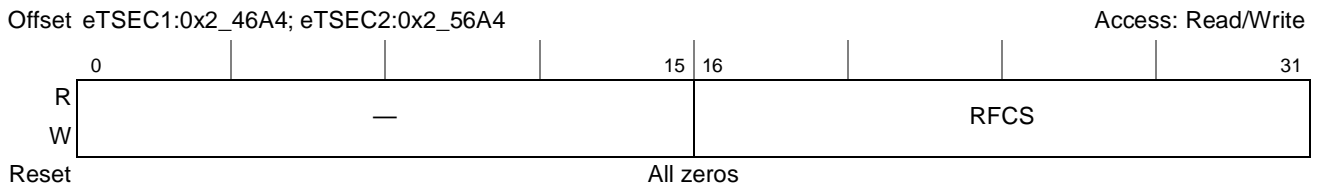
Table 16-63 describes the fields of the RPKT register.

**Table 16-63. 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).

### 16.5.3.6.10 Receive FCS Error Counter (RFCS)

Figure 16-60 describes the definition for the RFCS register.



**Figure 16-60. Receive FCS Error Counter Register Definition**

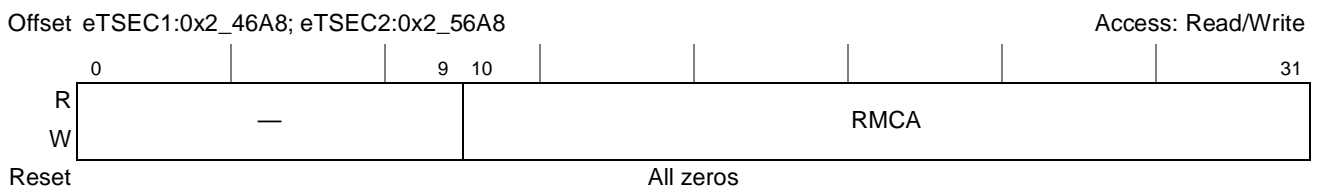
Table 16-64 describes the fields of the RFCS register.

**Table 16-64. 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.

### 16.5.3.6.11 Receive Multicast Packet Counter (RMCA)

Figure 16-61 describes the definition for the RMCA register.



**Figure 16-61. Receive Multicast Packet Counter Register Definition**

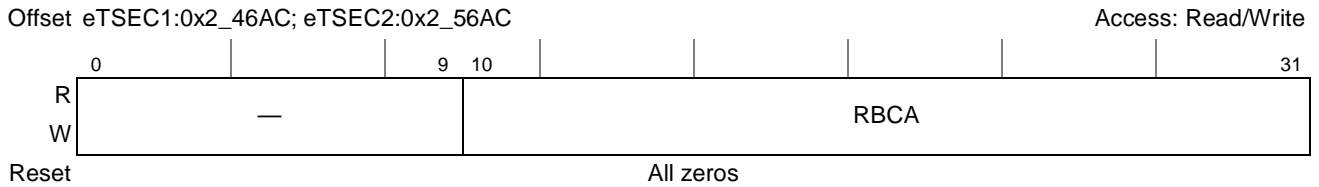
Table 16-65 describes the fields of the RMCA register.

**Table 16-65. 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.

### 16.5.3.6.12 Receive Broadcast Packet Counter (RBCA)

Figure 16-62 describes the definition for the RBCA register.



**Figure 16-62. Receive Broadcast Packet Counter Register Definition**

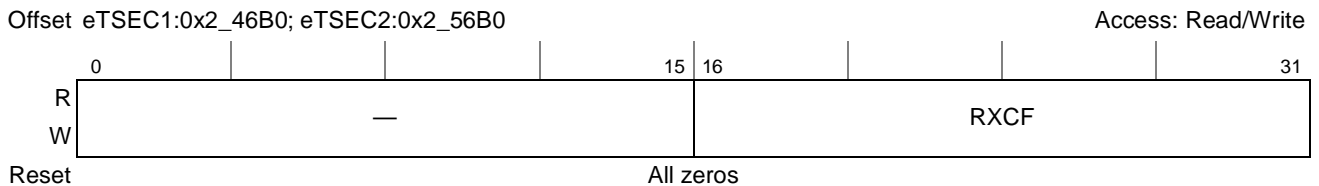
Table 16-66 describes the fields of the RBCA register.

**Table 16-66. 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.

### 16.5.3.6.13 Receive Control Frame Packet Counter (RXCF)

Figure 16-63 describes the definition for the RXCF register.



**Figure 16-63. Receive Control Frame Packet Counter Register Definition**

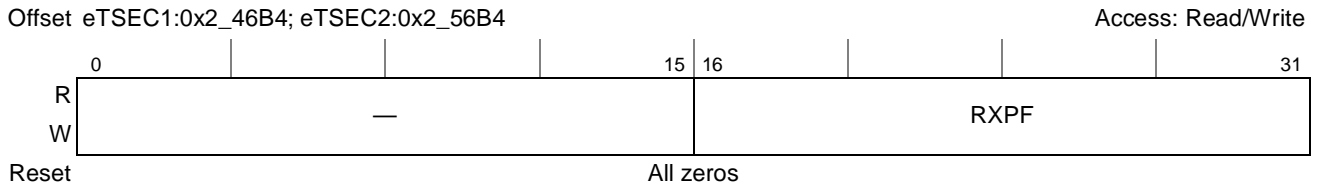
Table 16-67 describes the fields of the RXCF register.

**Table 16-67. 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).

### 16.5.3.6.14 Receive Pause Frame Packet Counter (RXPF)

Figure 16-64 describes the definition for the RXPF register.



**Figure 16-64. Receive Pause Frame Packet Counter Register Definition**

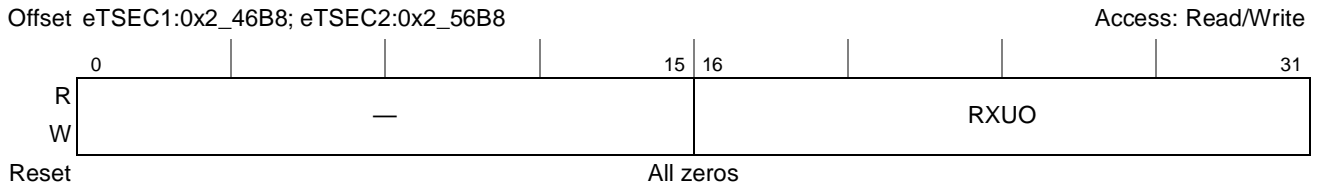
Table 16-68 describes the fields of the RXPF register.

**Table 16-68. 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).

### 16.5.3.6.15 Receive Unknown Opcode Packet Counter (RXUO)

Figure 16-65 describes the definition for the RXUO register.



**Figure 16-65. Receive Unknown Opcode Packet Counter Register Definition**

Table 16-69 describes the fields of the RXUO register.

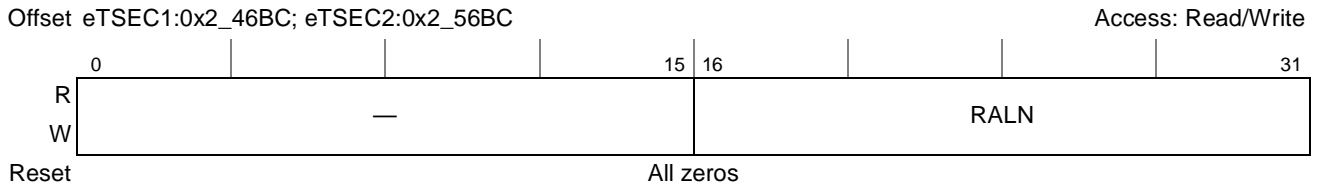
**Table 16-69. RXUO Field Descriptions**

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).



### 16.5.3.6.16 Receive Alignment Error Counter (RALN)

Figure 16-66 describes the definition for the RALN register.



**Figure 16-66. Receive Alignment Error Counter Register Definition**

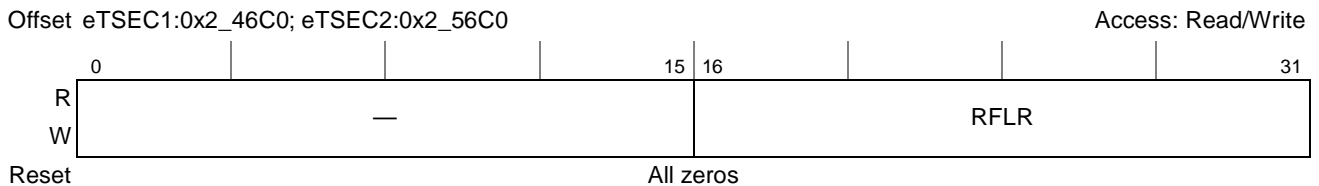
Table 16-70 describes the fields of the RALN register.

**Table 16-70. 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.

### 16.5.3.6.17 Receive Frame Length Error Counter (RFLR)

Figure 16-67 describes the definition for the RFLR register.



**Figure 16-67. Receive Frame Length Error Counter Register Definition**

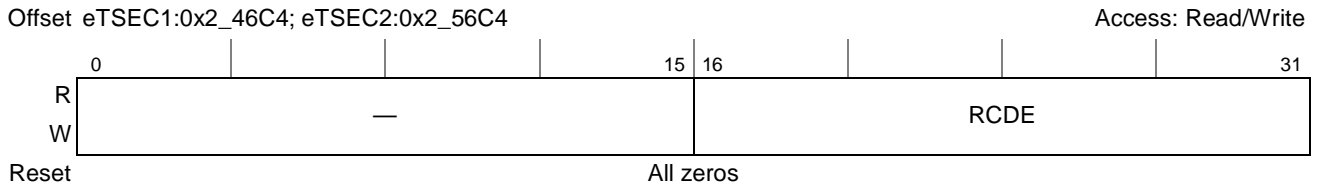
Table 16-71 describes the fields of the RFLR register.

**Table 16-71. 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.

### 16.5.3.6.18 Receive Code Error Counter (RCDE)

Figure 16-68 describes the definition for the RCDE register.



**Figure 16-68. Receive Code Error Counter Register Definition**

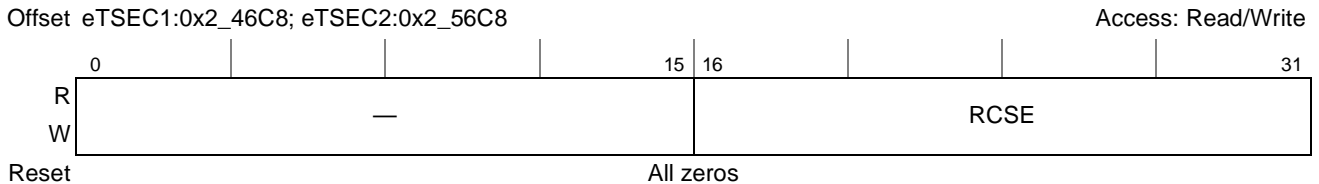
Table 16-72 describes the fields of the RCDE register.

**Table 16-72. 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.

### 16.5.3.6.19 Receive Carrier Sense Error Counter (RCSE)

Figure 16-69 describes the definition for the RCSE register.



**Figure 16-69. Receive Carrier Sense Error Counter Register Definition**

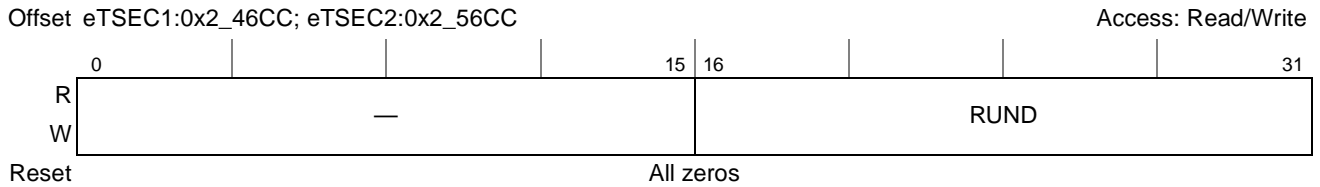
Table 16-73 describes the fields of the RCSE register.

**Table 16-73. 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. 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 <sub>n</sub> _RX_ER and an 0xE on TSEC <sub>n</sub> _RXD. Only one false carrier condition can be detected and logged between frames.

### 16.5.3.6.20 Receive Undersize Packet Counter (RUND)

Figure 16-70 describes the definition for the RUND register.



**Figure 16-70. Receive Undersize Packet Counter Register Definition**

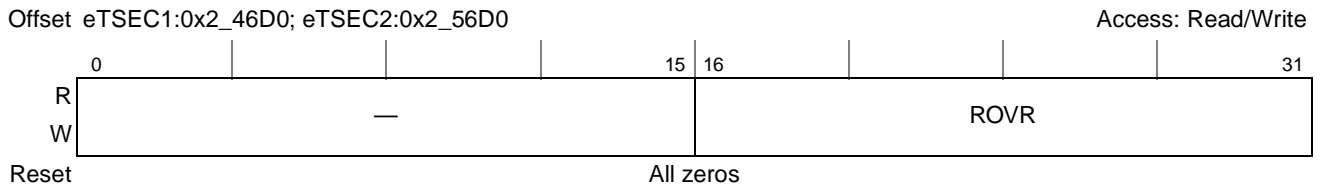
Table 16-74 describes the fields of the RUND register.

**Table 16-74. 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.

### 16.5.3.6.21 Receive Oversize Packet Counter (ROVR)

Figure 16-71 describes the definition for the ROVR register.



**Figure 16-71. Receive Oversize Packet Counter Register Definition**

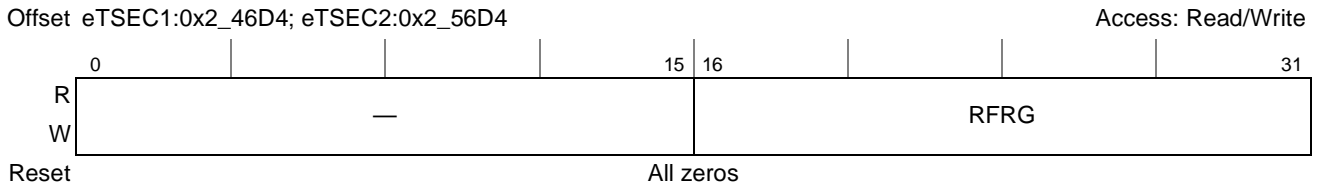
Table 16-75 describes the fields of the ROVR register.

**Table 16-75. 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.

### 16.5.3.6.22 Receive Fragments Counter (RFRG)

Figure 16-72 describes the definition for the RFRG register.



**Figure 16-72. Receive Fragments Counter Register Definition**

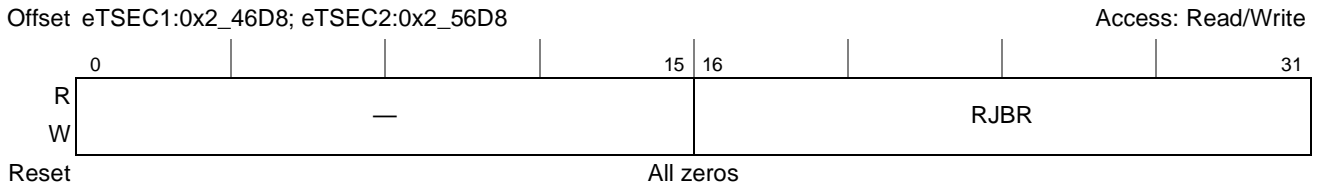
Table 16-76 describes the fields of the RFRG register.

**Table 16-76. 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.

### 16.5.3.6.23 Receive Jabber Counter (RJBR)

Figure 16-73 describes the definition for the RJBR register.



**Figure 16-73. Receive Jabber Counter Register Definition**

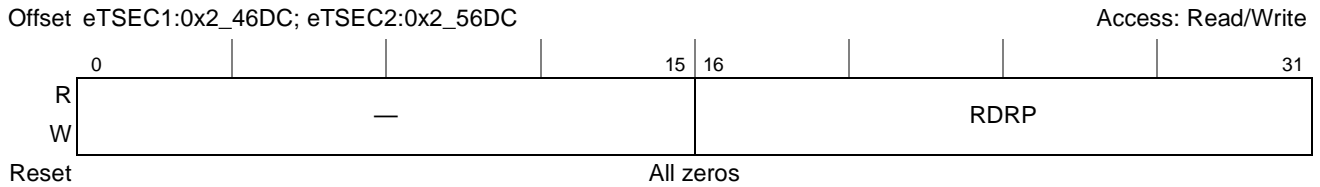
Table 16-77 describes the fields of the RJBR register.

**Table 16-77. 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.

### 16.5.3.6.24 Receive Dropped Packet Counter (RDRP)

Figure 16-74 describes the definition for the RDRP register.



**Figure 16-74. Receive Dropped Packet Counter Register Definition**

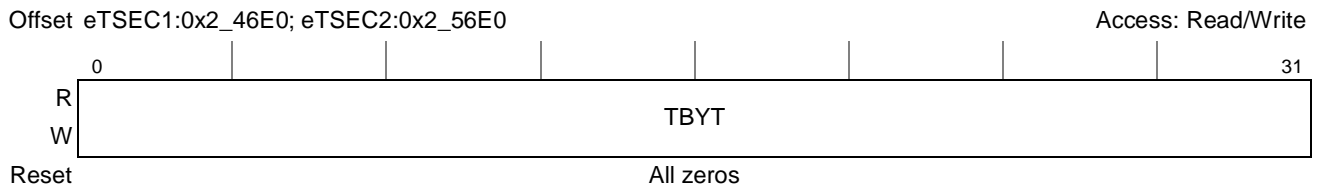
Table 16-78 describes the fields of the RDRP register.

**Table 16-78. 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.

### 16.5.3.6.25 Transmit Byte Counter (TBYT)

Figure 16-75 depicts the TBYT register.



**Figure 16-75. Transmit Byte Counter Register Definition**

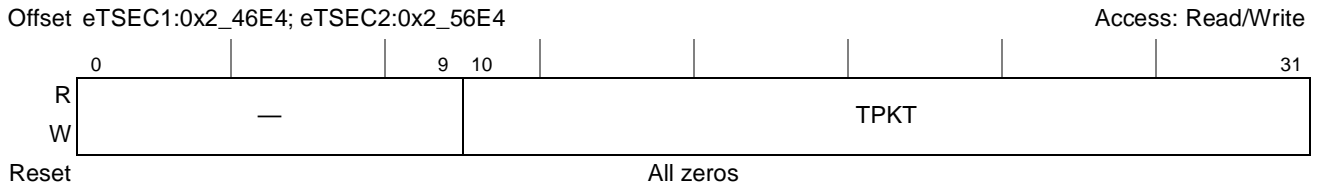
Table 16-79 describes the fields of the TBYT register.

**Table 16-79. 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. 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.

### 16.5.3.6.26 Transmit Packet Counter (TPKT)

Figure 16-76 describes the definition for the TPKT register.



**Figure 16-76. Transmit Packet Counter Register Definition**

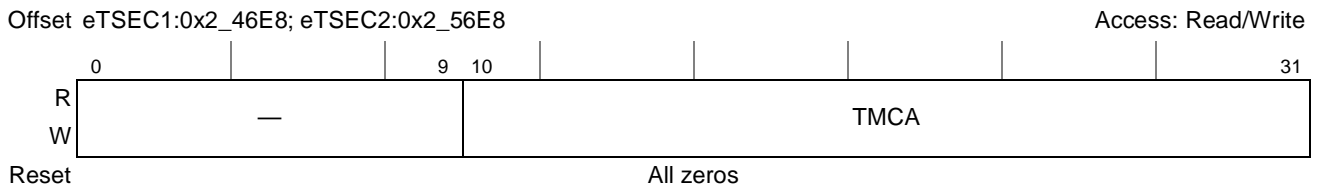
Table 16-80 describes the fields of the TPKT register.

**Table 16-80. 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).

### 16.5.3.6.27 Transmit Multicast Packet Counter (TMCA)

Figure 16-77 describes the definition for the TMCA register.



**Figure 16-77. Transmit Multicast Packet Counter Register Definition**

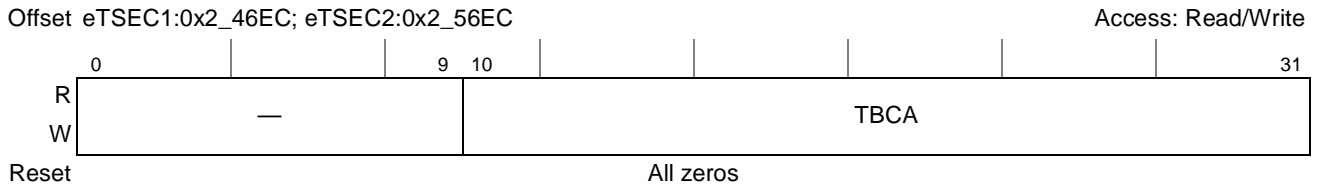
Table 16-81 describes the fields of the TMCA register.

**Table 16-81. 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).

### 16.5.3.6.28 Transmit Broadcast Packet Counter (TBCA)

Figure 16-78 describes the definition for the TBCA register.



**Figure 16-78. Transmit Broadcast Packet Counter Register Definition**

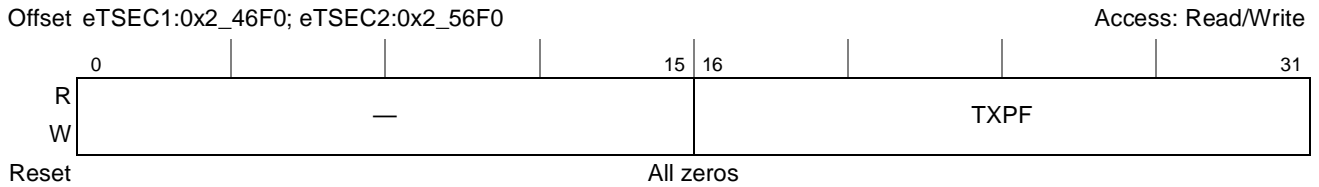
Table 16-82 describes the fields of the TBCA register.

**Table 16-82. 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).

### 16.5.3.6.29 Transmit Pause Control Frame Counter (TXPF)

Figure 16-79 describes the definition for the TXPF register.



**Figure 16-79. Transmit Pause Control Frame Counter Register Definition**

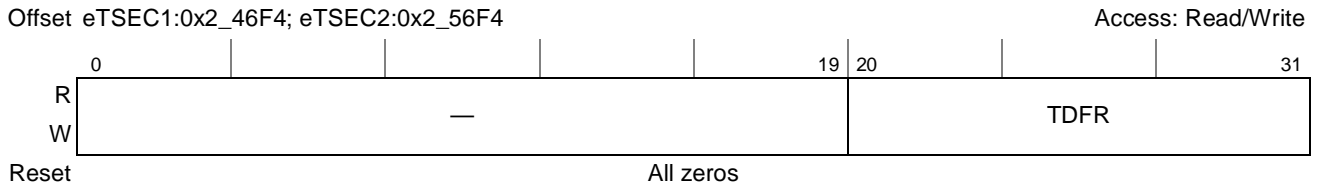
Table 16-83 describes the fields of the TXPF register.

**Table 16-83. 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).

### 16.5.3.6.30 Transmit Deferral Packet Counter (TDFR)

Figure 16-80 describes the definition for the TDFR register.



**Figure 16-80. Transmit Deferral Packet Counter Register Definition**

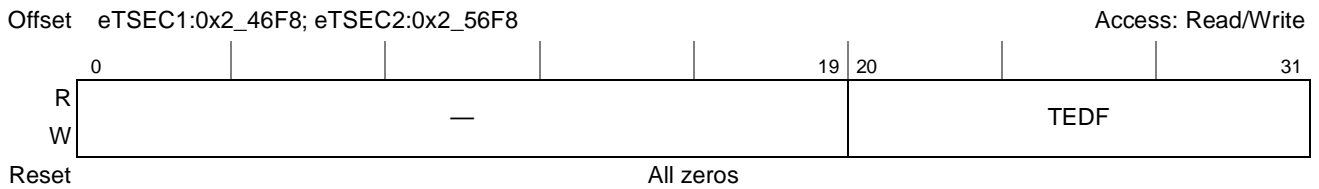
Table 16-84 describes the fields of the TDFR register.

**Table 16-84. 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.

### 16.5.3.6.31 Transmit Excessive Deferral Packet Counter (TEDF)

Figure 16-81 describes the definition for the TEDF register.



**Figure 16-81. Transmit Excessive Deferral Packet Counter Register Definition**

Table 16-85 describes the fields of the TEDF register.

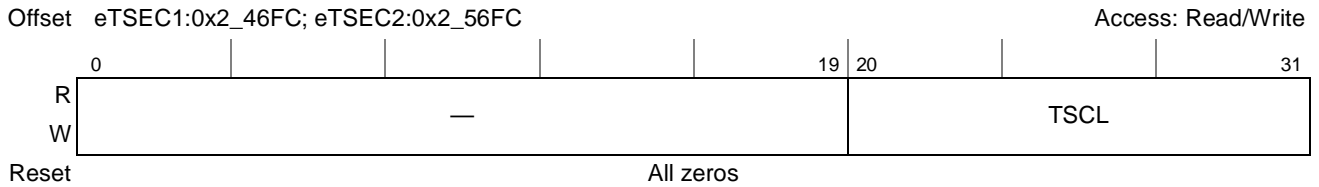
**Table 16-85. TEDF Field Descriptions**

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).



### 16.5.3.6.32 Transmit Single Collision Packet Counter (TSCL)

Figure 16-82 describes the definition for the TSCL register.



**Figure 16-82. Transmit Single Collision Packet Counter Register Definition**

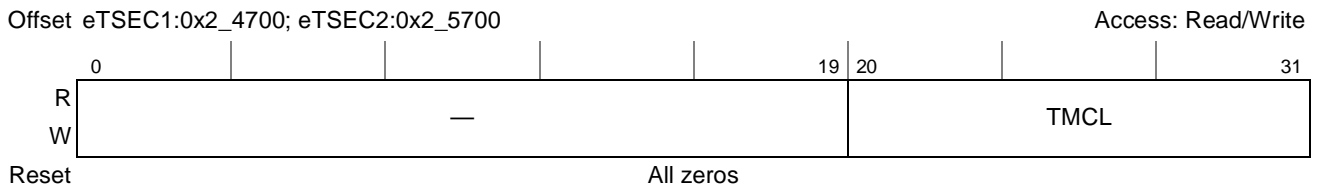
Table 16-86 describes the fields of the TSCL register.

**Table 16-86. 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.

### 16.5.3.6.33 Transmit Multiple Collision Packet Counter (TMCL)

Figure 16-83 describes the definition for the TMCL register.



**Figure 16-83. Transmit Multiple Collision Packet Counter Register Definition**

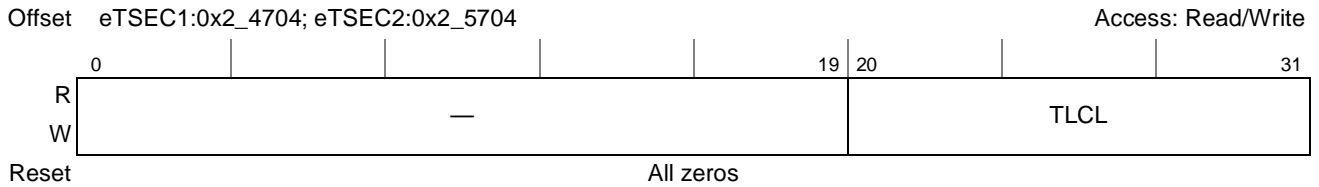
Table 16-87 describes the fields of the TMCL register.

**Table 16-87. 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.

### 16.5.3.6.34 Transmit Late Collision Packet Counter (TLCL)

Figure 16-84 describes the definition for the TLCL register.



**Figure 16-84. Transmit Late Collision Packet Counter Register Definition**

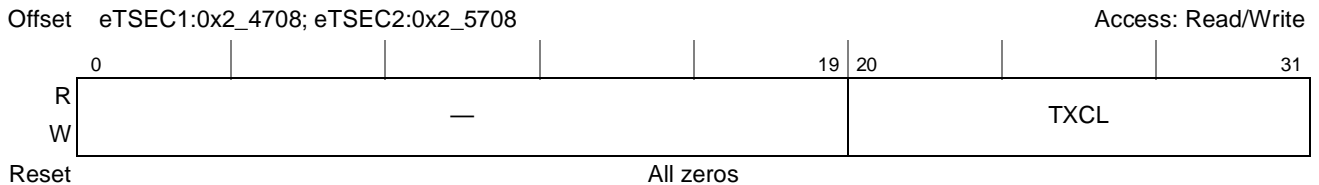
Table 16-88 describes the fields of the TLCL register.

**Table 16-88. 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.

### 16.5.3.6.35 Transmit Excessive Collision Packet Counter (TXCL)

Figure 16-85 describes the definition for the TXCL register.



**Figure 16-85. Transmit Excessive Collision Packet Counter Register Definition**

Table 16-89 describes the fields of the TXCL register.

**Table 16-89. 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.

### 16.5.3.6.36 Transmit Total Collision Counter (TNCL)

Figure 16-86 describes the definition for the TNCL register.

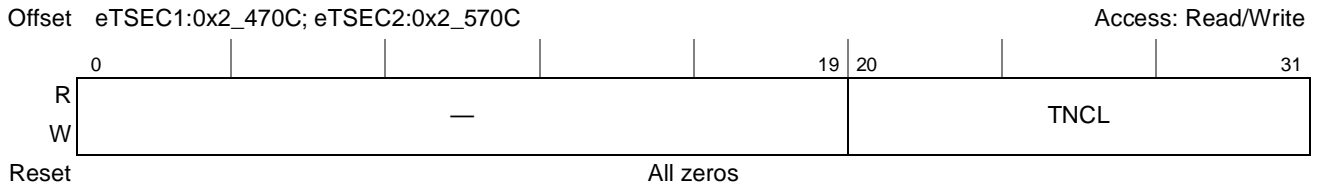


Figure 16-86. Transmit Total Collision Counter Register Definition

Table 16-90 describes the fields of the TNCL register.

Table 16-90. 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). <b>Note:</b> This count does not include collisions that result in an excessive collision condition.

### 16.5.3.6.37 Transmit Drop Frame Counter (TDRP)

Figure 16-87 describes the definition for the TDRP register.

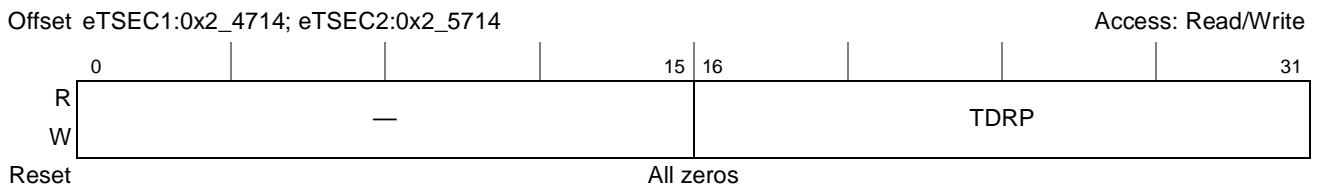


Figure 16-87. Transmit Drop Frame Counter Register Definition

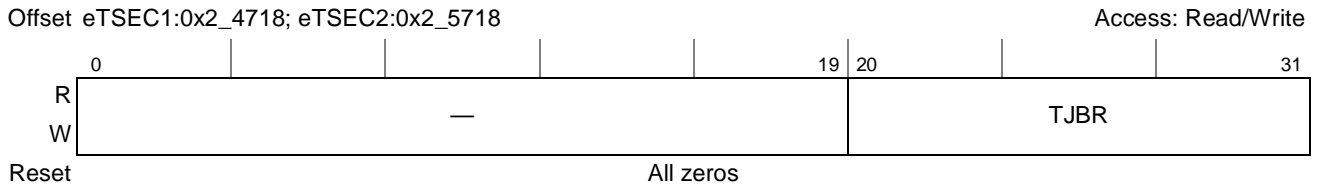
Table 16-91 describes the fields of the TDRP register.

Table 16-91. 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.

### 16.5.3.6.38 Transmit Jabber Frame Counter (TJBR)

Figure 16-88 describes the definition for the TJBR register.



**Figure 16-88. Transmit Jabber Frame Counter Register Definition**

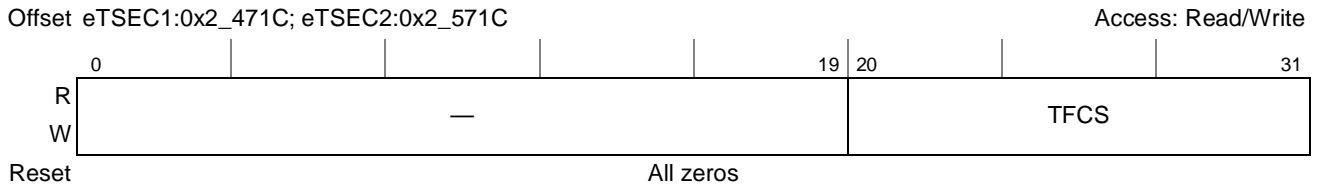
Table 16-92 describes the fields of the TJBR register.

**Table 16-92. 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.

### 16.5.3.6.39 Transmit FCS Error Counter (TFCS)

Figure 16-89 describes the definition for the TFCS register.



**Figure 16-89. Transmit FCS Error Counter Register Definition**

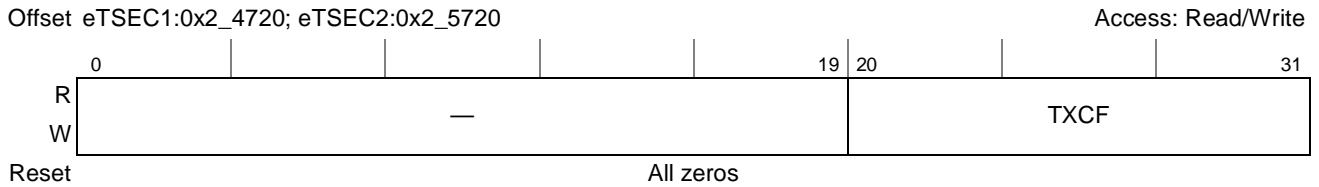
Table 16-93 describes the fields of the TFCS register.

**Table 16-93. 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.

### 16.5.3.6.40 Transmit Control Frame Counter (TXCF)

Figure 16-90 describes the definition for the TXCF register.



**Figure 16-90. Transmit Control Frame Counter Register Definition**

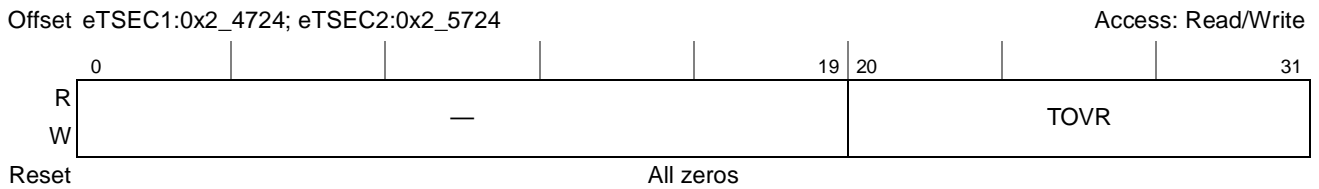
Table 16-94 describes the fields of the TXCF register.

**Table 16-94. 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).

### 16.5.3.6.41 Transmit Oversize Frame Counter (TOVR)

Figure 16-91 describes the definition for the TOVR register.



**Figure 16-91. Transmit Oversized Frame Counter Register Definition**

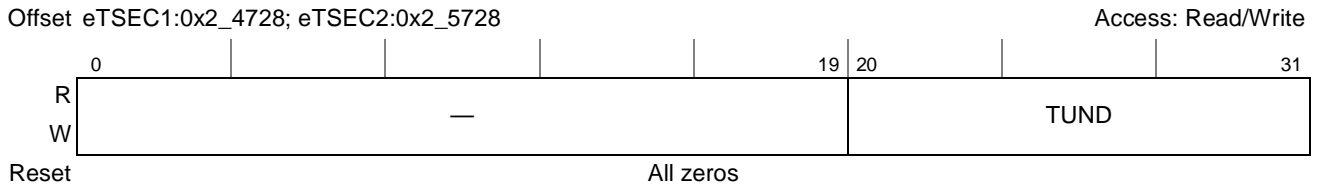
Table 16-95 describes the fields of the TOVR register.

**Table 16-95. 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 (non VLAN) or 1522 (VLAN) with a correct FCS value.

### 16.5.3.6.42 Transmit Undersize Frame Counter (TUND)

Figure 16-92 describes the definition for the TUND register.



**Figure 16-92. Transmit Undersize Frame Counter Register Definition**

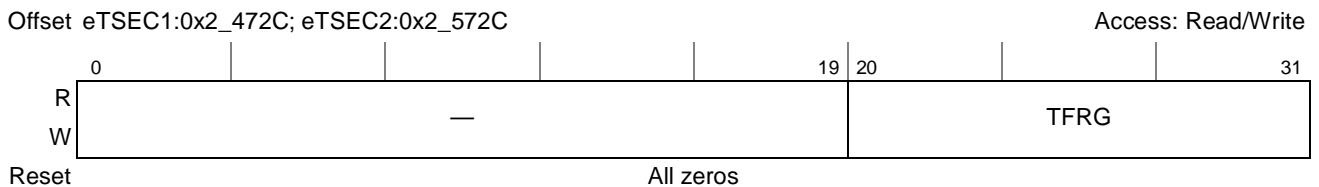
Table 16-96 describes the fields of the TUND register.

**Table 16-96. TUND Field Descriptions**

Bits	Name	Description
0–19	—	Reserved
20–31	TUND	Transmit undersize frame counter. Increments for every frame less than 64 bytes, with a correct FCS value.

### 16.5.3.6.43 Transmit Fragment Counter (TFRG)

Figure 16-93 describes the definition for the TFRG register.



**Figure 16-93. Transmit Fragment Counter Register Definition**

Table 16-97 describes the fields of the TFRG register.

**Table 16-97. TFRG Field Descriptions**

Bits	Name	Description
0–19	—	Reserved
20–31	TFRG	Transmit fragment counter. Increments for every frame less than 64 bytes, with an incorrect FCS value.

### 16.5.3.6.44 Carry Register 1 (CAR1)

Carry register bits are cleared on carry register writes when the respective bits are set. Figure 16-94 describes the definition for the CAR1 register.

Offset eTSEC1:0x2\_4730; eTSEC2:0x2\_5730

Access: w1c

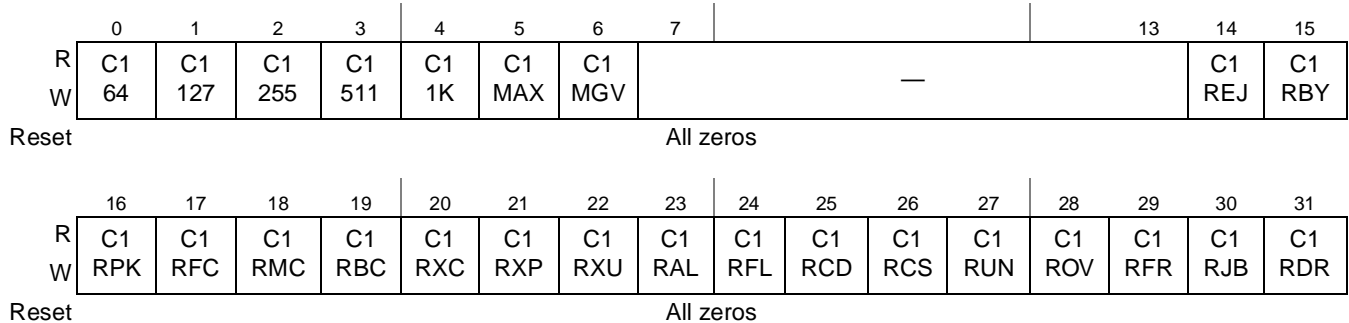


Figure 16-94. Carry Register 1 (CAR1) Register Definition

Table 16-98 describes the fields of the CAR1 register.

Table 16-98. 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
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

**Table 16-98. CAR1 Field Descriptions (continued)**

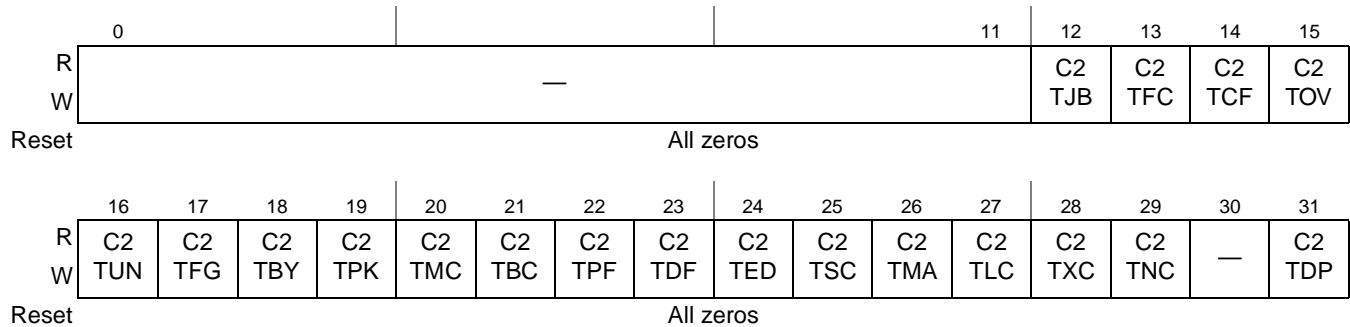
Bits	Name	Description
25	C1RCD	Carry register 1 RCDE counter carry bit
26	C1RCS	Carry register 1 RCSE counter carry bit
27	C1RUN	Carry register 1 RUND counter carry bit
28	C1ROV	Carry register 1 ROVR counter carry bit
29	C1RFR	Carry register 1 RFRG counter carry bit
30	C1RJB	Carry register 1 RJBR counter carry bit
31	C1RDR	Carry register 1 RDRP counter carry bit

**16.5.3.6.45 Carry Register 2 (CAR2)**

Figure 16-95 describes the definition for the CAR2 register.

Offset eTSEC1:0x2\_4734; eTSEC2:0x2\_5734

Access: w1c

**Figure 16-95. Carry Register 2 (CAR2) Register Definition**

Carry register bits are cleared on carry register write when the respective bits are set. Table 16-99 describes the fields of the CAR2 register.

**Table 16-99. 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



**Table 16-99. CAR2 Field Descriptions (continued)**

Bits	Name	Description
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
30	—	Reserved, should be cleared
31	C2TDP	Carry register 2 TDRP counter carry bit

#### 16.5.3.6.46 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 16-96](#) describes the definition for the CAM1 register.

Offset eTSEC1:0x2\_4738; eTSEC2:0x2\_5738

Access: Read/Write

	0	1	2	3	4	5	6	7					13	14	15	
R	M1	M1	M1	M1	M1	M1	M1		—					M1	M1	
W	64	127	255	511	1K	MAX	MGV							REJ	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	M1	M1	M1	M1	M1	M1	M1	M1	M1	M1	M1	M1	M1	M1	M1	M1
W	RPK	RFC	RMC	RBC	RXC	RXP	R XU	RAL	RFL	RCD	RCS	RUN	ROV	RFR	RJB	RDR
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

**Figure 16-96. Carry Mask Register 1 (CAM1) Register Definition**

[Table 16-100](#) describes the fields of the CAM1 register.

**Table 16-100. 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

Table 16-100. CAM1 Field Descriptions (continued)

Bits	Name	Description
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
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

### 16.5.3.6.47 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 16-97 describes the definition for the CAM2 register.

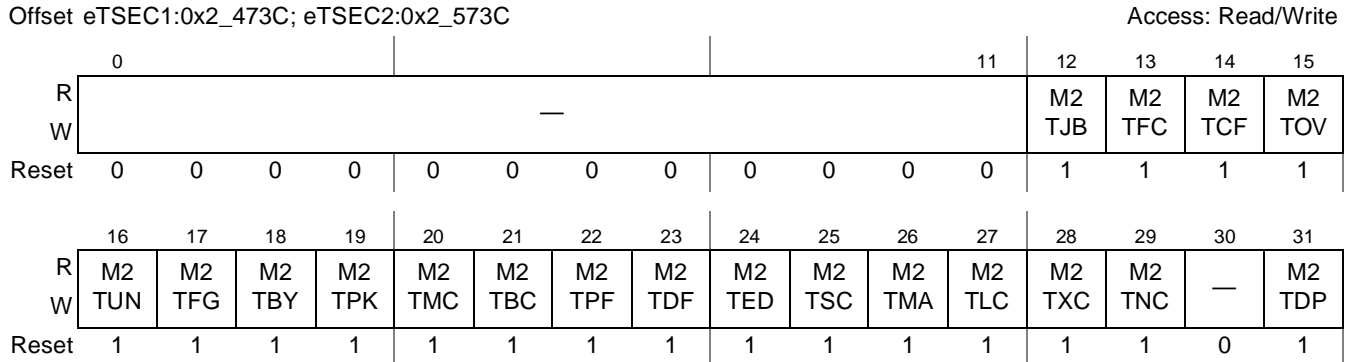


Figure 16-97. Carry Mask Register 2 (CAM2) Register Definition

Table 16-101 describes the fields of the CAM2 register.

Table 16-101. CAM2 Field Descriptions

Bits	Name	Description
0–11	—	Reserved
12	M2TJB	Mask register 2 TJBR counter carry bit mask
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

**Table 16-101. CAM2 Field Descriptions (continued)**

Bits	Name	Description
30	—	Reserved
31	M2TDP	Mask register 2 TDRP counter carry bit mask

### 16.5.3.6.48 Receive Filer Rejected Packet Counter (RREJ)

Figure 16-98 describes the definition for the RREJ register.

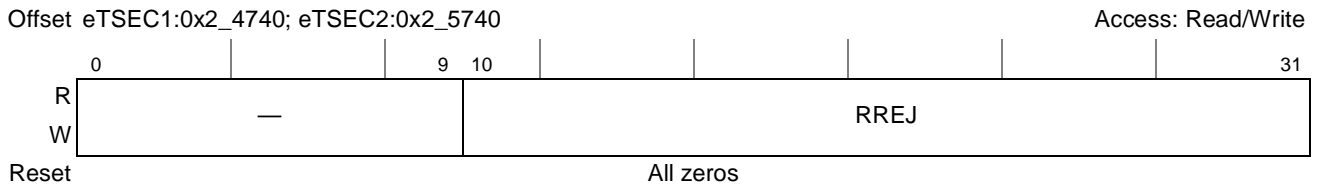
**Figure 16-98. Receive Filer Rejected Packet Counter Register Definition**

Table 16-102 describes the fields of the RREJ register.

**Table 16-102. 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 RxBDRing that was not enabled (see IEVENT[FIQ] error).

### 16.5.3.7 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.

For more information on the hash algorithm, see [Section 16.6.2.7.2, “Hash Table Algorithm.”](#)

### 16.5.3.7.1 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 16-99 describes the definition for the IGADDR $n$  register.

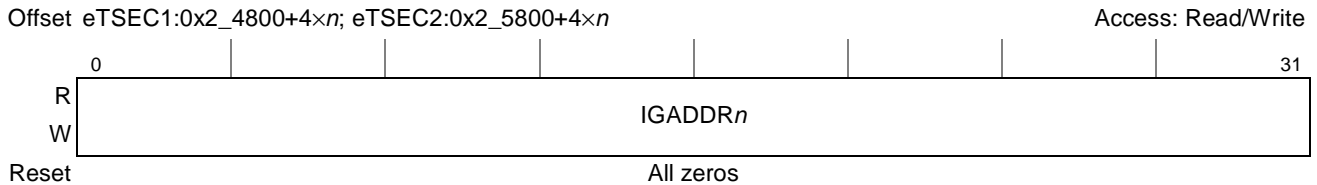


Figure 16-99. IGADDR $n$  Register Definition

Table 16-103 describes the fields of the IGADDR $n$  register.

Table 16-103. 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.

### 16.5.3.7.2 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 16-100 describes the definition for the GADDR $n$  register.

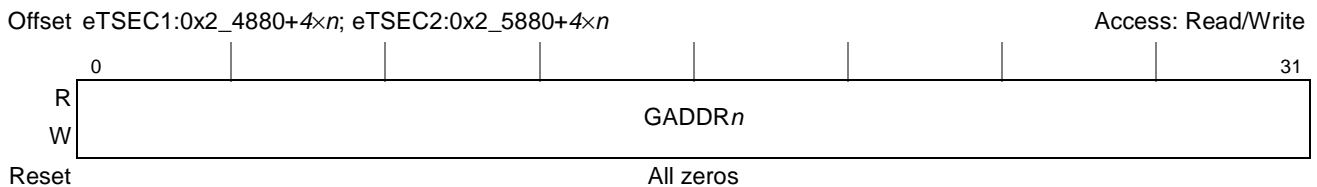


Figure 16-100. GADDR $n$  Register Definition

Table 16-104 describes the fields of the GADDR $n$  register.

**Table 16-104. 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.

### 16.5.3.8 DMA Attribute Registers

This section describes the two eTSEC DMA attribute registers.

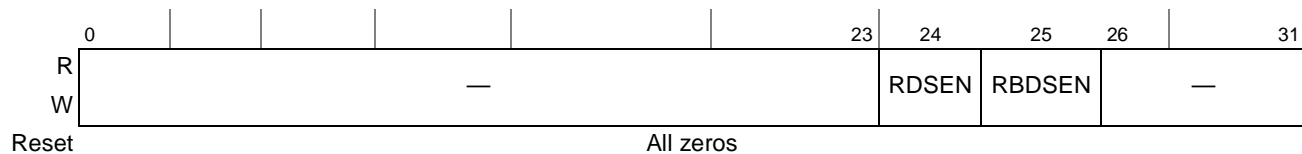
#### 16.5.3.8.1 Attribute Register (ATTR)

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.

Figure 16-101 describes the definition for the ATTR register.

Offset eTSEC1:0x2\_4BF8; eTSEC2:0x2\_5BF8

Access: Read/Write



**Figure 16-101. ATTR Register Definition**

Table 16-105 describes the fields of the ATTR register.

**Table 16-105. ATTR Field Descriptions**

Bits	Name	Description
0–23	—	Reserved
24	RDSSEN	Rx data snoop enable. 0 Disables snooping of all receive frames data to memory. 1 Enables snooping of all receive frames data to memory.
25	RBDSSEN	RxBD snoop enable. 0 Disables snooping of all receive BD memory accesses. 1 Enables snooping of all receive BD memory accesses.
26–31	—	Reserved

#### 16.5.3.8.2 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 16-102 describes the definition for the ATTRELI register.

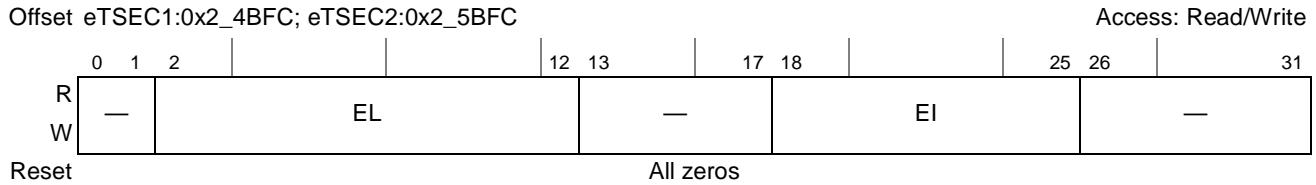


Figure 16-102. ATTRELI Register Definition

Table 16-106 describes the fields of the ATTRELI register.

Table 16-106. 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.

### 16.5.3.9 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 16.6.5.1/16-154 for the theory of operation of these registers.

#### 16.5.3.9.1 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 16-103 describes the definition for the RQPRM $n$  register.

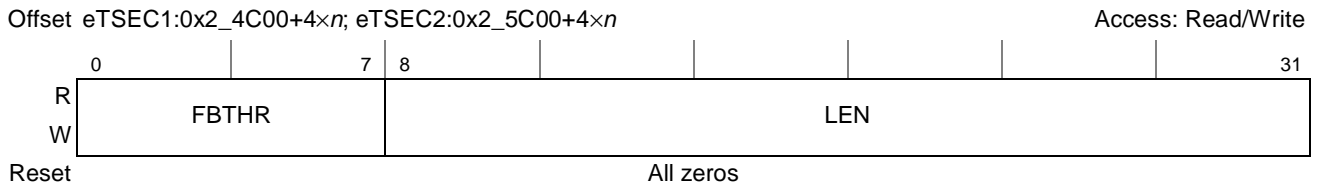


Figure 16-103. RQPRM Register Definition

Table 16-107 describes the fields of the RQPRM register.

Table 16-107. 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.

### 16.5.3.9.2 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 32-bit 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 256-Mbyte 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 16.6.5.1, “Back Pressure Determination through Free Buffers.” Figure 16-104 describes the definition for the RFBPTR $n$  register.

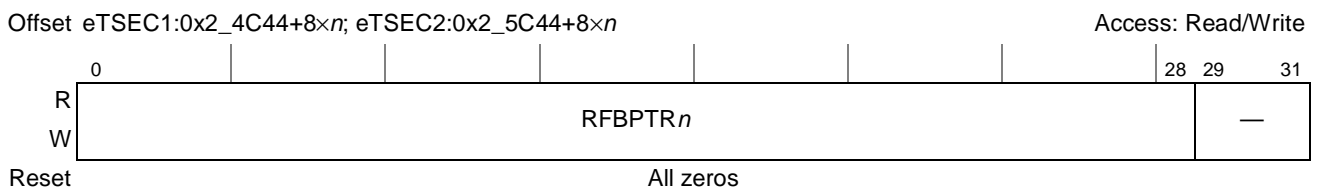


Figure 16-104. RFBPTR0–RFBPTR7 Register Definition



Table 16-108 describes the fields of the RFBPTR $n$  registers.

**Table 16-108. 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 $n$ . 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.

### 16.5.3.10 IEEE 1588-Compatible Timestamping Registers

IEEE 1588 compliant timestamping on this device is accomplished using the per-port transmit timestamping registers within each Ethernet controller memory space (See Section 16.5.3.2.10, “Transmit Time Stamp Identification Register (TMR\_TXTS1–2\_ID),” and Section 16.5.3.2.11, “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.

#### 16.5.3.10.1 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 16-7 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	ALM1P	ALM2P	—	FS	PP1L	PP2L	TCLK_PERIOD										
W																	
Reset	0	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	RTPE	FRD	—	ESFDP	ESFDE	ETEP2	ETEP1	COPH	CIPH	TMSR	—	BYP	TE	CKSEL			
W																	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	

**Figure 16-105. TMR\_CTRL Register Definition**

Table 16-109 describes the fields of the TMR\_CTRL register. Register fields not described below are reserved.

**Table 16-109. TMR\_CTRL Register Field Descriptions**

Bits	Name	Description
0	ALM1P	Alarm1 output polarity 0 active high output 1 active low output
1	ALM2P	Alarm2 output polarity 0 active high output 1 active low output
2	—	Reserved
3	FS	FIPER start indication 0 Fiper is enabled through timer enable 1 Fiper is enabled through timer enable and alarm indication.
4	PP1L	Fiper1 pulse loopback mode enabled. 0 Trigger1 input is based upon normal external trigger input. 1 Fiper1 pulse is looped back into Trigger1 input.
5	PP2L	Fiper2 pulse loopback mode enabled. 0 Trigger2 input is based upon normal external trigger input. 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. For nanosecond granularity on 1588 timer counter rate, the TCLK_PERIOD should be calculated using the following equation: $TCLK\_PERIOD = 10^9 / \text{Nominal\_Frequency}$
16	RTPE	Record Tx Time-Stamp to PAL Enable. 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 <a href="#">Section 16.6.6.5, “Time-Stamp Insertion on Transmit Packets.”</a>
17	FRD	FIPER Realignment Disable 0 Fiper Realignment is enabled. 1 Fiper Realignment is disabled.
18–19	—	Reserved
20	ESFDP	External Tx/Rx SFD Polarity. 0 Time stamp on rising edge of external SFD indication. 1 Time stamp on falling edge of external SFD indication.
21	ESFDE	External Tx/Rx SFD Enable. 0 Time stamp PTP TX frame based on MAC's SFD indication. 1 Time stamp PTP TX frame based on external SFD indication from PHY.
22	ETEP2	External trigger 2 edge polarity 0 Time stamp on the rising edge of the external trigger 1 Time stamp on the falling edge of the external trigger

Table 16-109. TMR\_CTRL Register Field Descriptions (continued)

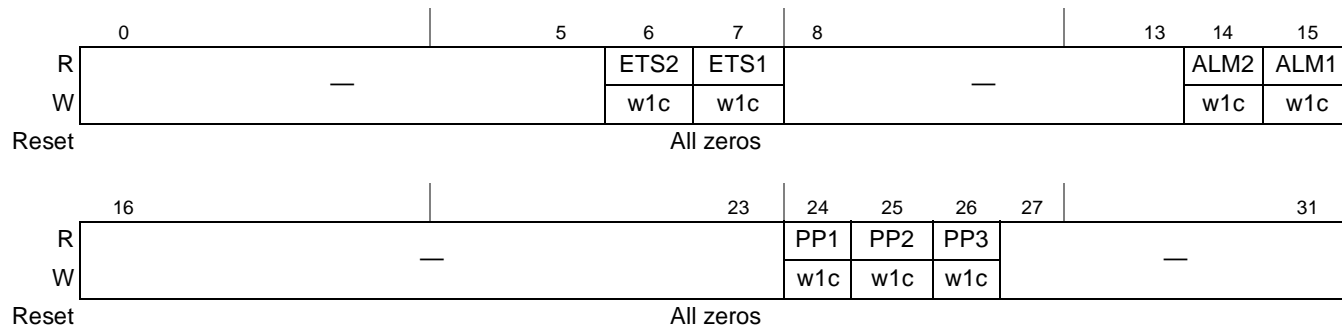
Bits	Name	Description
23	ETEP1	External trigger 1 edge polarity 0 time stamp on the rising edge of the external trigger 1 time stamp on the falling edge of the external trigger
24	COPH	Generated clock (TSEC_1588_GCLK) output phase. 0 non-inverted divided clock is output 1 inverted divided clock is output
25	CIPH	Oscillator input clock phase. 0 non-inverted timer input clock 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. 0 normal operation 1 place entire timer in reset except control and config registers NOTE: Prior to initiating timer reset (setting TMSR), must gracefully stop receiver (See MACCFG1[RX_EN] description). 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
28	BYP	Bypass drift compensated clock 0 64-bit clock counter is incremented on the accumulator overflow 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. 0 timer not enabled 1 timer enabled and resume normal operation
30–31	CKSEL	1588 Timer reference clock source select. 00 External high precision timer reference clock (TSEC_TMR_CLK) 01 eTSEC system clock 10 Reserved 11 RTC clock input Note that the 1588 reference clock must be no slower than 1/7 the Rx_clk frequency. 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.

### 16.5.3.10.2 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 16-4 describes the definition for the TMR\_TEVENT register.

Offset eTSEC1:0x2\_4E04

Access: w1c



**Figure 16-106. TMR\_TEVENT Register Definition**

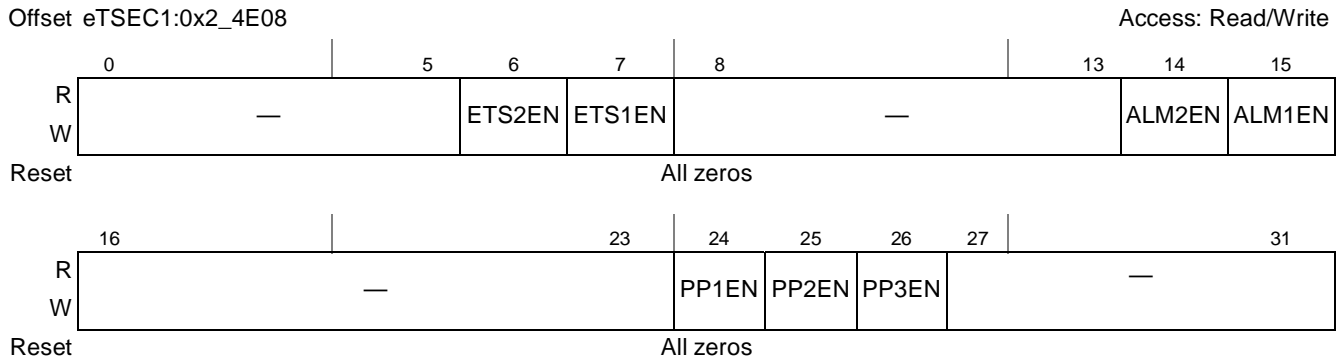
Table 16-110 describes the fields of the TMR\_TEVENT register fields for the timer.

**Table 16-110. TMR\_TEVENT Register Field Descriptions**

Bits	Name	Description
0–5	—	Reserved
6	ETS2	External trigger 2 timestamp sampled 0 external trigger timestamp not sampled 1 external trigger timestamp sampled
7	ETS1	External trigger 1 timestamp sampled 0 external trigger timestamp not sampled 1 external trigger timestamp sampled
8–13	—	Reserved
14	ALM2	Current time equaled alarm time register 2 0 alarm time has not be reached yet 1 alarm time has been reached
15	ALM1	Current time equaled alarm time register 1 0 alarm time has not be reached yet 1 alarm time has been reached
16–23	—	Reserved
24	PP1	Indicates that a periodic pulse has been generated based on FIPER1 register. 0 periodic pulse not generated 1 periodic pulse generated
25	PP2	Indicates that a periodic pulse has been generated based on FIPER2 register. 0 periodic pulse not generated 1 periodic pulse generated
26	PP3	Indicates that a periodic pulse has been generated based on FIPER3 register. 0 periodic pulse not generated 1 periodic pulse generated
27–31	—	Reserved

### 16.5.3.10.3 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 16-111 describes the definition for the TMR\_TEMASK register.



**Table 16-111. TMR\_TEMASK Register Definition**

Table 16-112 describes the fields of the TMR\_TEMASK register fields for the timer.

**Table 16-112. 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	PP3EN	Periodic pulse event 3 enable
27–31	—	Reserved

### 16.5.3.10.4 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 16-107 describes the definition for the TMR\_PEVENT register.

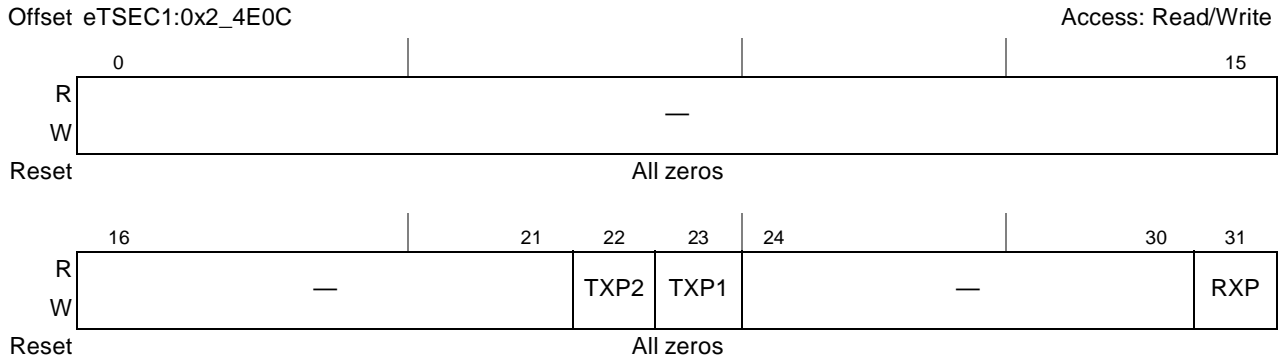


Figure 16-107. TMR\_PEVENT Register Definition

Table 16-113 describes the fields of the TMR\_PEVENT register fields for the timer.

Table 16-113. 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. 0 PTP packet not transmitted 1 PTP packet has been transmitted
23	TXP1	Indicates that a PTP frame has been transmitted and its timestamp is stored in TXTS1 register. 0 PTP packet not transmitted 1 PTP packet has been transmitted
24–30	—	Reserved
31	RXP	Indicates that a PTP frame has been received 0 PTP packet not received 1 PTP packet has been received

### 16.5.3.10.5 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 16-108 describes the definition for the TMR\_PEMASK register.

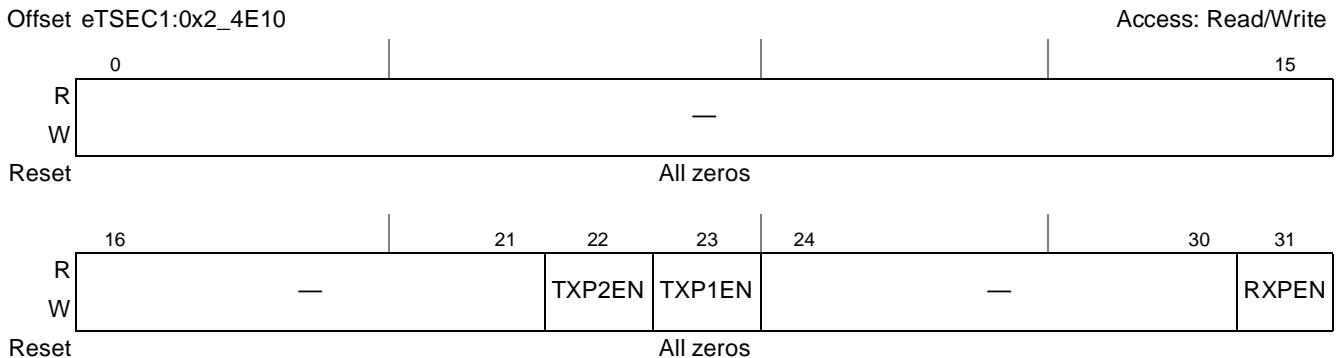


Figure 16-108. TMR\_PEMASK Register Definition

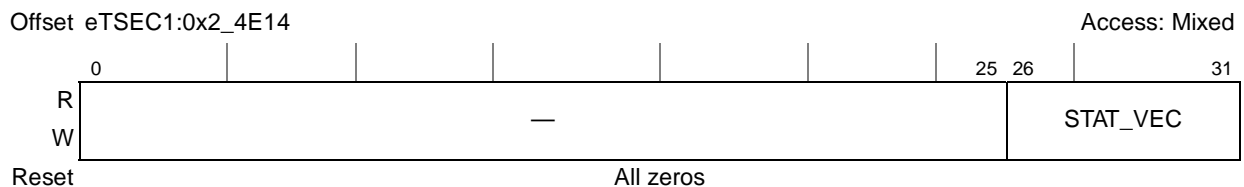
Table 16-114 describes the fields of the TMR\_PEMASK register fields for the timer.

**Table 16-114. 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

### 16.5.3.10.6 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 16-115 describes the definition for the TMR\_STAT register.



**Table 16-115. TMR\_STAT Register Definition**

Table 16-116 describes the fields of the TMR\_STAT register.

**Table 16-116. 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.

### 16.5.3.10.7 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 16-109 describes the definition for the TMR\_CNT\_H/L register.

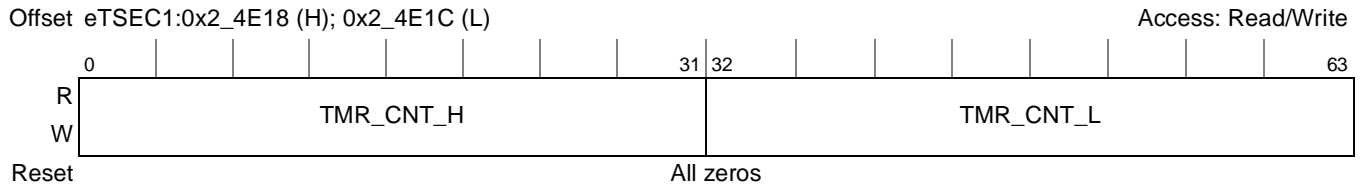


Figure 16-109. TMR\_CNT\_H Register Definition

Table 16-117 describes the fields of the TMR\_CNT\_H/L register.

Table 16-117. TMR\_CNT\_H/L Register Field Descriptions

Bits	Name	Description
0–63	TMR_CNT_H/L	<p>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.</p> <p>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.</p>

### 16.5.3.10.8 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  $2^{32}/\text{FreqDivRatio}$ . 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 16-110 describes the definition of the TMR\_ADD register.

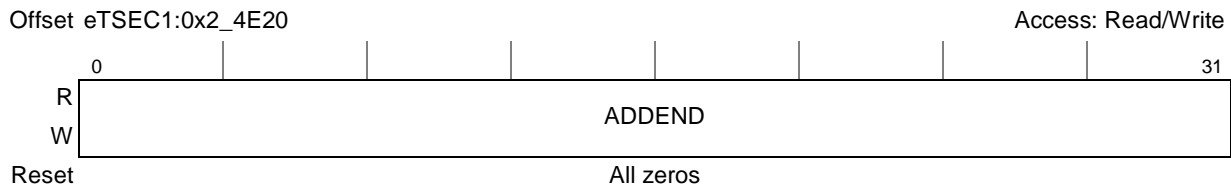


Figure 16-110. TMR\_ADD Register Definition



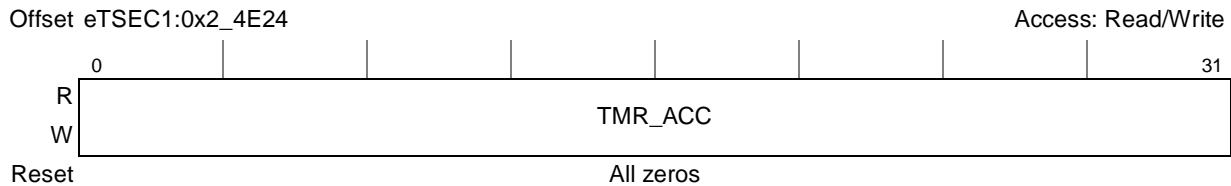
Table 16-118 describes the fields of the TMR\_ADD register fields for the timer.

**Table 16-118. TMR\_ADD Register Field Descriptions**

Bits	Name	Description
0–31	ADDEND	Timer drift compensation addend register value. It is programmed with a value of $2^{32}/\text{FreqDivRatio}$ . For example, TimerOsc = 50 MHz NominalFreq = 40 MHz FreqDivRatio = 1.25 ADDEND = $\text{ceil}(2^{32}/1.25) = 0xCCCC\_CCCD$

### 16.5.3.10.9 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 16-111 describes the definition of the TMR\_ACC register.



**Figure 16-111. TMR\_ACC Register Definition**

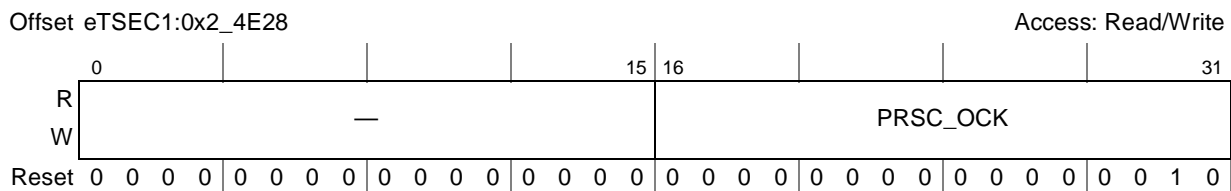
Table 16-119 describes the fields of the TMR\_ACC register.

**Table 16-119. TMR\_ACC Register Field Descriptions**

Bits	Name	Description
0–31	TMR_ACC	32-bit timer accumulator register

### 16.5.3.10.10 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 16-112 describes the definition for the TMR\_PRSC register.



**Figure 16-112. TMR\_PRSC Register Definition**



Table 16-122 describes the fields of the TMR\_ALARM $n$ \_H/L register.

**Table 16-122. 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. 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. In FS mode the alarm trigger is used as an indication to the fiber start down counting. Only alarm 1 supports this mode. In FS mode, alarm polarity bit should be configured to 0 (rising edge).

### 16.5.3.10.13 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.

$$\text{FIPER\_VALUE} = (\text{prescale\_value} \times \text{tclk\_per} \times N) - \text{tclk\_per}$$

For example:

$$\text{prescale} = 9$$

$$\text{clock period} = 10$$

The FIPER can get the following values: 80, 170, 260 .....

The three registers in eTSEC1 are shared for all eTSECs. Figure 16-115 describes the definition for the TMR\_FIPER register.

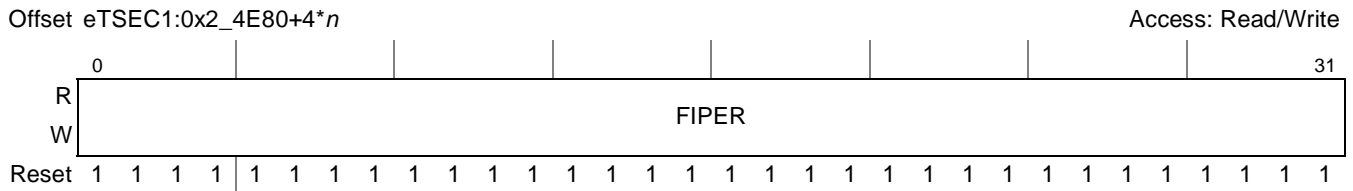


Figure 16-115. TMR\_FIPER $n$  Register Definition

Table 16-123 describes the fields of the TMR\_FIPER register.

Table 16-123. 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.

#### 16.5.3.10.14 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 16-116 describes the definition for the TMR\_ETTS $n$ \_H/L register.

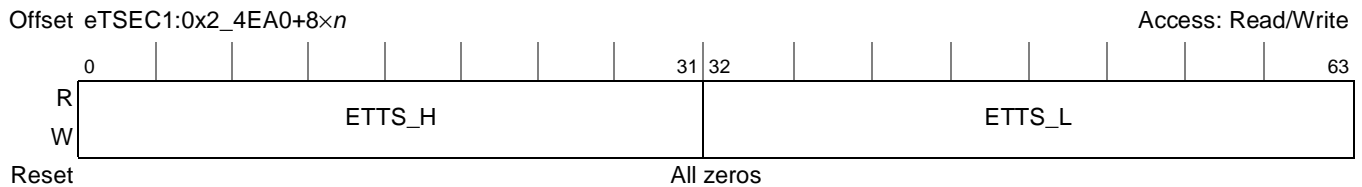


Figure 16-116. TMR\_ETTS1-2\_H/L Register Definition

Table 16-124 describes the fields of the TMR\_ETTS $n$ \_H/L register.

Table 16-124. 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.

## 16.6 Functional Description

This section discusses the following:

- [Section 16.6.1, “Connecting to Physical Interfaces on Ethernet”](#)
- [Section 16.6.2, “Gigabit Ethernet Controller Channel Operation”](#)
- [Section 16.6.3, “TCP/IP Off-Load”](#)
- [Section 16.6.4, “Quality of Service \(QoS\) Provision”](#)

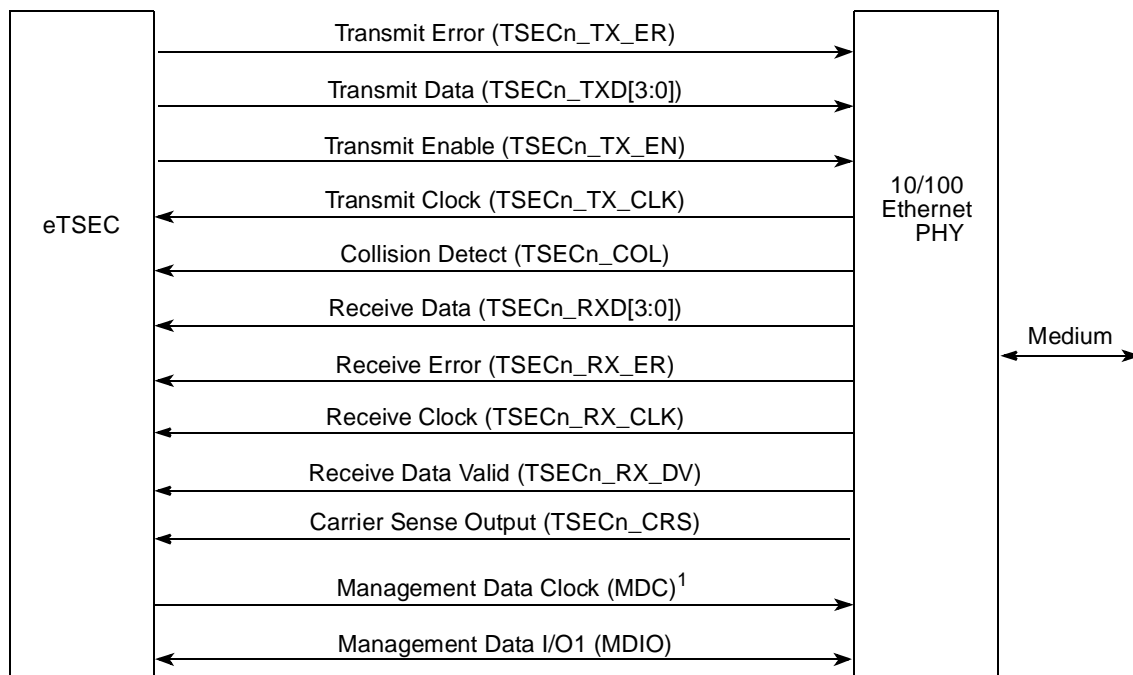
- Section 16.6.5, “Lossless Flow Control”
- Section 16.6.7, “Buffer Descriptors”

## 16.6.1 Connecting to Physical Interfaces on Ethernet

This section describes how to connect the eTSEC to various interfaces: MII and RGMII. 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<sub>n</sub>\_TXD[3:0], bit 3 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.

### 16.6.1.1 Media-Independent Interface (MII)

This section describes the media-independent interface (MII) intended to be used between the PHYs and the eTSEC. Figure 16-117 depicts the basic components of the MII, including the signals required to establish eTSEC module connection with a PHY.



<sup>1</sup> 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.

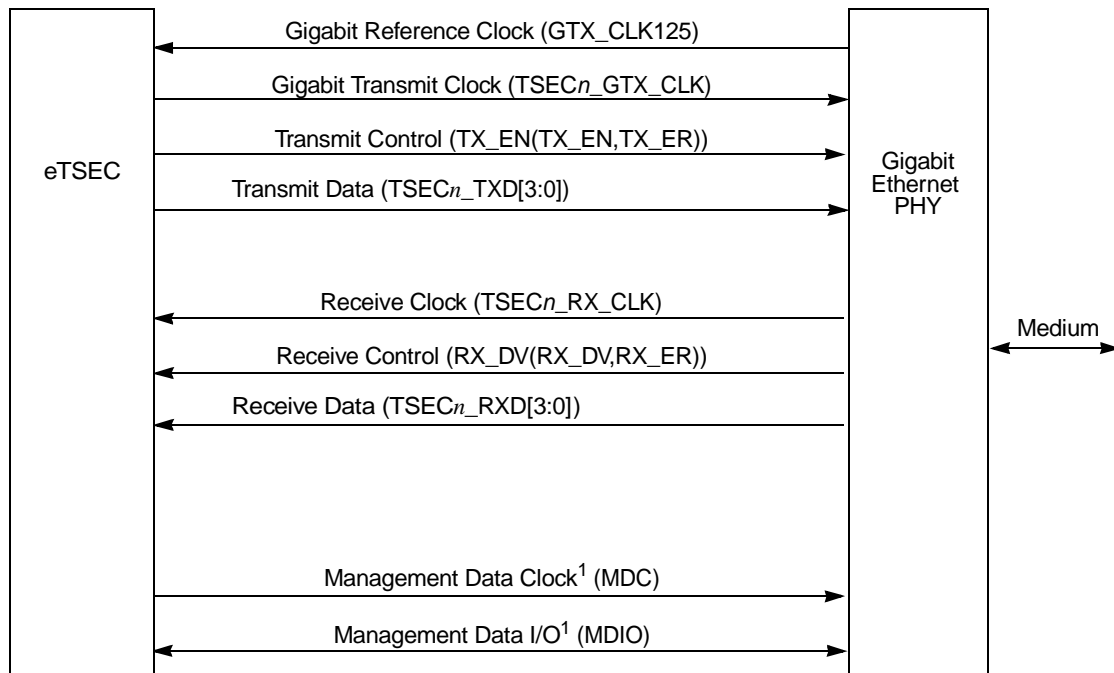
**Figure 16-117. eTSEC-MII Connection**

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.

### 16.6.1.2 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

IEEE 802.3z GMII. 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 125 MHz, 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 16-118 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.



<sup>1</sup> 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 16-118. eTSEC-RGMII Connection**

### 16.6.1.3 Ethernet Physical Interfaces Signal Summary

Table 16-125 describes the signal multiplexing for MII interfaces.

**Table 16-125. RGMII and MII Signals Multiplexing**

RGMII Interface			MII Interface		
Frequency [MHz] 125			Frequency [MHz] 25		
Voltage[V] 3.3/2.5			Voltage[V] 3.3		
Signals (TSEC <sub>n</sub> )	I/O	No. of Signals	Signals (TSEC <sub>n</sub> )	I/O	No. of Signals
GTX_CLK	O	1	TX_CLK	I	1
GTX_CLK125	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
TX_EN/TX_ER	O	1	TX_EN	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
RX_DV/RX_ER	I	1	RX_DV	I	1
			RX_ER	I	1
COL	not used		COL	I	1
CRS	not used		CRS	I	1
<b>Sum</b>		13	<b>Sum</b>		16

Table 16-126 describes the signals shared by all interfaces.

**Table 16-126. Shared Signals**

Signals	I/O	No. of Signals	Function
MDIO	I/O	1	Management interface I/O
MDC	O	1	Management interface clock
<b>Sum</b>		2	—

## 16.6.2 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.

### 16.6.2.1 Initialization Sequence

This section describes which registers are reset due to a hard or software reset and what registers the user must initialize prior to enabling the eTSEC.

#### 16.6.2.1.1 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 II Pro 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.

#### 16.6.2.1.2 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 16-3](#) for the register list. [Table 16-127](#) describes the minimum steps for register initialization.

**Table 16-127. 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 MII/RGMII
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 16.6.7.1, “Data Buffer Descriptors,”](#) for more information.

### 16.6.2.2 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 TBASE0–TBASE7 with new Tx BD pointers
6. Load 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 filter table (through RQFAR, RQFCR, and RQFPR) if filtering 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

### 16.6.2.3 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 MII/RGMII 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 10/100 Mbps mode it sends bytes least-significant nibble first and each nibble is sent lsb first. While it is in 1000 Mbps mode it sends bytes LSB first.

#### 16.6.2.4 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 TSECn\_RX\_DV is asserted and as long as TSECn\_COL remains negated (full-duplex mode ignores TSECn\_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 begins to perform the frame recognition function through destination address (DA) recognition (see [Section 16.6.2.7, “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 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.

### 16.6.2.5 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 100/1000 Mbps speeds.

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 . 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.

#### 16.6.2.5.1 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 16-119](#).

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 16-119. Definition of Custom Preamble Sequence**

The fields of the custom preamble sequence are described in [Table 16-128](#). 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 16-128. 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.

### 16.6.2.5.2 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 16-120](#).

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 16-120. Definition of Received Preamble Sequence**

The fields of the received preamble sequence are described in [Table 16-129](#). 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.

**Table 16-129. 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

### 16.6.2.6 RMON Support

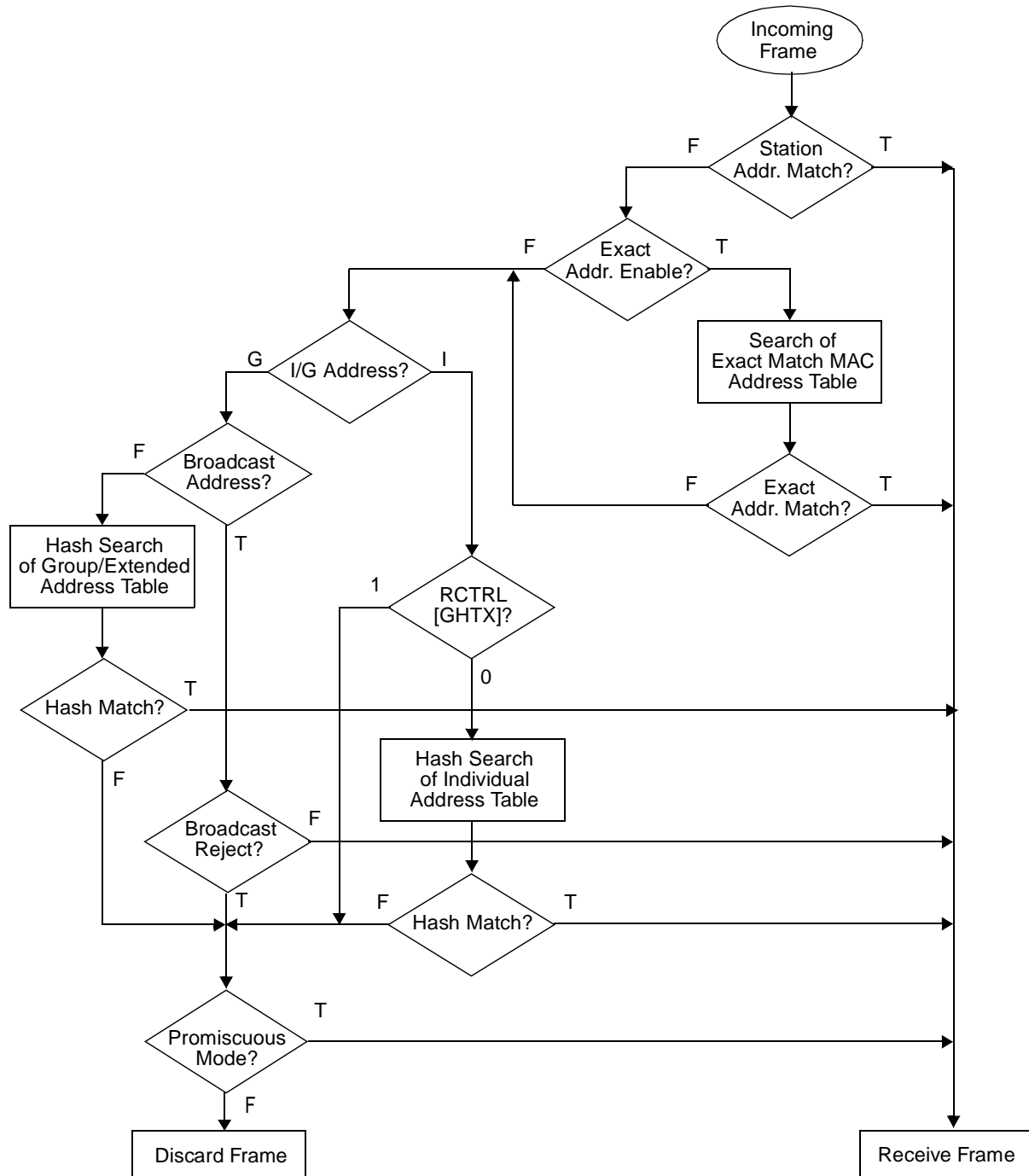
Using promiscuous mode, the eTSEC can automatically gather network statistics required for remote network interface monitoring. The RMON MIB group 1, RMON MIB group 2, RMON MIB group 3, RMON MIB group 9, RMON MIB2, and the IEEE 802.3 Ethernet MIB are supported. For RMON statistics and their corresponding counters, see the memory map.

### 16.6.2.7 Frame Recognition

The Ethernet controller performs frame recognition using destination address (DA) recognition. A frame can be rejected or accepted based on the outcome.

### 16.6.2.7.1 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 16-121](#).



**Figure 16-121. 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 filter. Therefore even in

promiscuous mode it remains possible to program the filter 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 MACxADDR1 and MACxADDR2 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.

### 16.6.2.7.2 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 0xFFFF\_FFFF 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 shown below. 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.

The following code snippet shows sample C code for computing eTSEC hash table indices.

---

```

/* 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;
}

```

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 filter may be used to reject frames with unintended address hits in the hash table.

## 16.6.2.8 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 16-130 lists the flow-control frame structure.

**Table 16-130. 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
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.

## 16.6.2.9 Interrupt Handling

The following describes what usually occurs within an 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 RxBBD rings if IEVENT[RXC, RXB or RXF] is set. Consult register bits RSTAT[RXF0–RXF7] to determine which RxBBD 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 RxBBD; thus, it is important to check more than just one RxBBD during interrupt handling. Typically, all RxBBDs 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 16-131 describes the non-error transmit interrupts.

**Table 16-131. 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 16-132 shows the non-error receive interrupts.

**Table 16-132. 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 RxBBD rings.	Programmable 'write with response' RxBBD to memory before setting IEVENT[RXB].
RXF	Receive frame: A frame was received to an enabled RxBBD ring and the last receive buffer descriptor (RxBBD) of that frame was updated.	Programmable 'write with response' to memory on the last RxBBD before setting IEVENT[RXF].

### 16.6.2.9.1 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.

### 16.6.2.9.2 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.

### 16.6.2.9.3 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 16-133](#) 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 16-133. 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 $\mu$ s	1.68 s
0 (I/F clock)	100Base-T at 25 MHz	2.56 $\mu$ s	168 ms
0 (I/F clock)	1000Base-T at 125 MHz	0.51 $\mu$ s	33.6 ms
1 (sys. clock)	eTSEC operating at 125 MHz	0.512 $\mu$ s	33.5 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.

### 16.6.2.10 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 first TxBD pointer, TBPTR $n$ , of any given frame is located at a 16-byte aligned address.
- Each TxBD[Data Length] is greater-than or equal to 64 bytes.

If the first TxBD alignment restriction is not met, the back-to-back IPG may be as many as 32 cycles. 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).

### 16.6.2.11 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.

For loopback, TX\_CLK is required (from the Ethernet PHY) and RX\_CLK is not required. The output data appears on the TX pin.

### 16.6.2.12 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 16-134](#).

**Table 16-134. 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 16-135](#).

**Table 16-135. 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 RxB[OV], sets RxB[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.
Filed frame to invalid queue error	A frame is received and discarded as a result of the filer directing it to an RxB ring that is currently not enabled. The controller sets IEVENT[FIQ] and increments the discarded frame counter (RDRP).

**Table 16-135. Reception Errors (continued)**

Error	Description
Parser error	<p>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.</p> <p><b>Note:</b> 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.</p> <p>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.</p>
Non-octet error (dribbling bits)	<p>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.</p>
CRC error	<p>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.</p>
Memory read error	<p>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<math>n</math>] bit is set. The halted queue resumes reception once the RSTAT[QHLT<math>n</math>] bit is cleared.</p>
Data parity error	<p>Data in the receive FIFO or filer table was potentially corrupted. The controller sets IEVENT[DPE], but otherwise continues reception until halted explicitly.</p>
Babbling receive error	<p>A frame is received that exceeds the MAC's maximum frame length. The controller sets IEVENT[BABR] and continues.</p>

### 16.6.3 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.

### 16.6.3.1 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 16-122 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.

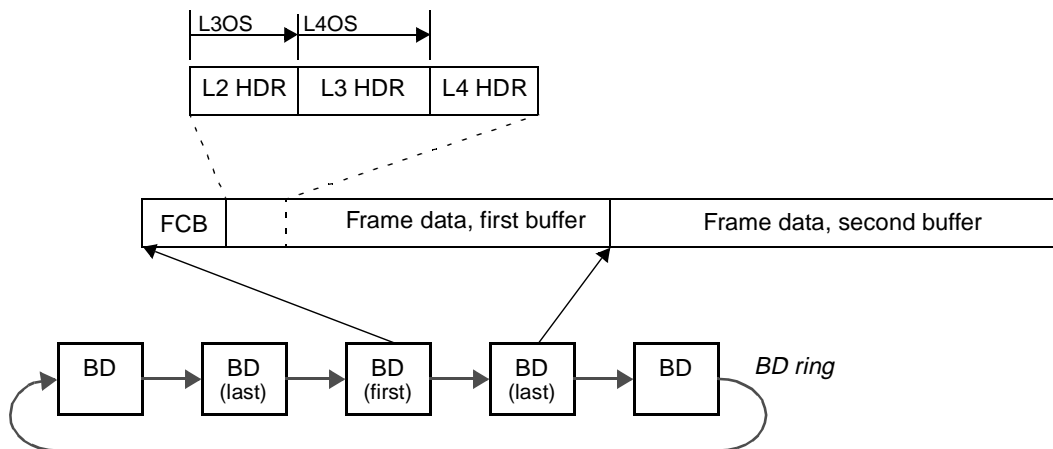


Figure 16-122. Location of Frame Control Blocks for TOE Parameters

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.



### 16.6.3.2 Transmit Path Off-Load and Tx PTP Packet Parsing

TOE functions for transmit are defined by the contents of the Tx FCB. [Figure 16-123](#) 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 16-123. 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 16-128](#).

The contents of the Tx FCB are defined in [Table 16-136](#).

**Table 16-136. 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. 0 Ignore VLCTL field. 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. 0 Ignore layer 3 and higher headers. 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. 0 IP header version is 4. 1 IP header version is 6.
	3	TUP	Layer 4 header is a TCP or UDP header. 0 Do not process any layer 4 header. 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. 0 Layer 4 protocol is either TCP (if TUP = 1) or undefined. 1 Layer 4 protocol is UDP if TUP = 1.

**Table 16-136. Tx Frame Control Block Description (continued)**

Bytes	Bits	Name	Description
0–1	5	CIP	Checksum IP header enable. 0 Do not generate an IP header checksum. 1 Generate an IPv4 header checksum.
	6	CTU	Checksum TCP or UDP header enable. 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. 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. 0 Calculate TCP or UDP pseudo-header checksum as normal, assuming that the IP header has no options. 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. 0 Do not attempt to capture transmission event time 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/ 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.

### 16.6.3.3 Receive Path Off-Load

Upon receive, the Rx FCB returns the status of frame parse and TOE functions applied to the accompanying frame. [Figure 16-124](#) 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		—	GPPF	
Offset + 2	RQ				PRO											
Offset + 4																
Offset + 6	VLCTL															

**Figure 16-124. Receive Frame Control Block**

The contents of the Rx FCB are defined in [Table 16-137](#).

**Table 16-137. 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. 0 No VLAN tag recognized. 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. 0 No layer 3 header recognized. 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.  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 <a href="#">Section 16.5.3.3.8, “Receive Queue Filer Table Property Register (RQFPR)”</a> ) indicates that the packet was fragmented.
	2	IP6	IP version 6 header found at layer 3. 0 No IPv6 header was found. 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. 0 No layer 4 header recognized. 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. 0 IPv4 header checksum not verified, either because verification was disabled or a valid IPv4 header could not be located. 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. 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. 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. 0 No checksum error in IPv4 header. 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. 0 No checksum error in TCP or UDP header. 1 Error in header checksum only if PRO = 0x06 or PRO = 0x11.
	8–11	—	Reserved
	12–13	PER	Parse error. 00 No error in L2 to L4 parse 01 Reserved 10 Inconsistent or unsupported L3 header sequence 11 Reserved
	14	—	Reserved
	15	GPFP	General-purpose filer event packet. This packet was filed based on matching a GPI rule sequence.

Table 16-137. 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 filer 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	<p>If IP = 1, PRO is set as follows:</p> <ul style="list-style-type: none"> <li>• PRO=0xFF for a fragment header or a back to back route header</li> <li>• PRO=0xnn for an unrecognized header, where nn is the next protocol field</li> <li>• PRO=(TCP/UDP header), as defined in the IANA specification, if TCP or UDP header is found</li> </ul> <p>If IP = 0, PRO is undefined.</p> <p>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] fields with whatever value was found in the protocol field of the IP header. See <a href="#">Section 16.5.3.3.8, "Receive Queue Filer Table Property Register (RQFPR),"</a> for a description of RQFPR.</p>
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.

## 16.6.4 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 them to the filer which essentially acts as a frame classifier.

### 16.6.4.1 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
- PPOES
- 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 16-138](#) describes what the Ethernet header parser recognizes for stack L2 header.

**Table 16-138. Supported Stack L2 Ethernet Headers**

Column—Current L2 Ethertype 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:

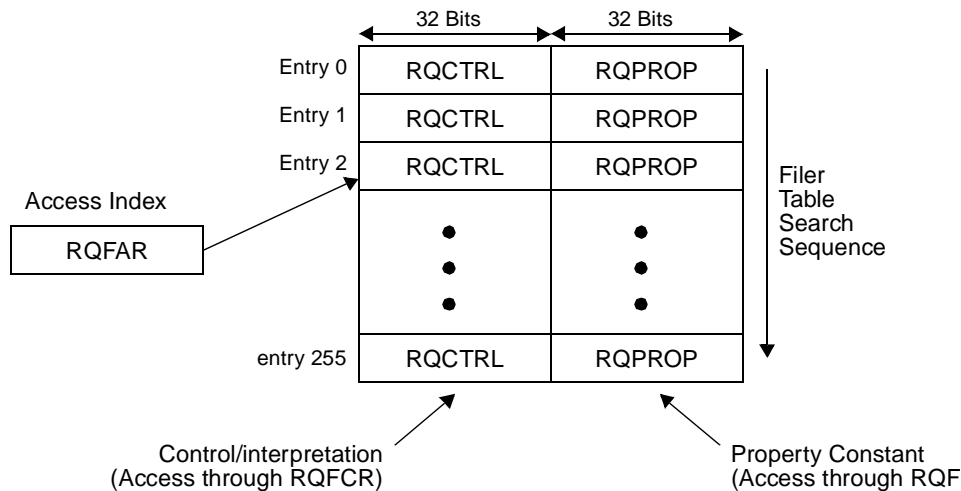
- IP recognition (v4/v6, ARP, encapsulated protocol)
- IP header checksum verification
- IPv4/6 over IPv4/6 (tunneling)—parse headers and find layer 4 protocol
- IP layer 4 protocol/next header extraction
- Stop parsing on unrecognized next header/protocol

- 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)

#### 16.6.4.2 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 16-125](#). 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 16-125. Structure of the Receive Queue Filer Table**

#### 16.6.4.2.1 Filing Rules

Unless the filer is disabled, every received frame from the Ethernet MAC 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 16.5.3.3.8, “Receive Queue Filer Table Property Register \(RQFPR\),”](#) on page 16-55 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.

#### 16.6.4.2.2 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.



### 16.6.4.2.3 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 16-139](#).

**Table 16-139. Special Filer Rules**

Rule Description	RQCTRL Fields						RQPROP Word	RQCTRL Word <sup>1</sup>
	CLE	REJ	AND	Q	CMP	PID		
Default file—Always file frame to ring Q	0	0	0	Q	01	0000	0x0000_0000	0x0000_0020
Default reject—Always discard frame	0	1	0	000_000	01	0000	0x0000_0000	0x0000_0120
Empty rule in AND—Always matches	0/1 <sup>2</sup>	0	1	000_000	01	0000	0xFFFF_FFFF	0x0000_00A0
Empty rule in rule set—Always fails	0/1 <sup>3</sup>	0	0	000_000	11	0000	0xFFFF_FFFF	0x0000_0060

<sup>1</sup> Hexadecimal digits *qq* denotes field Q shifted left 2 bits.

<sup>2</sup> Set CLE = 1 if the empty rule guards a cluster.

<sup>3</sup> Set CLE = 1 if the empty rule occurs at the end of a cluster.

### 16.6.4.2.4 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 16-140](#).

**Table 16-140. 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.

### 16.6.4.2.5 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 16-139](#). In many cases empty entries can be overwritten by software without stopping eTSEC's receive function.

### 16.6.4.2.6 Filer Example—802.1p Priority Filing

This example, shown in [Table 16-141](#), 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 16-141. 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 16-141. 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

#### 16.6.4.2.7 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 16-142. 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 16-142. 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

#### 16.6.4.2.8 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 16-143. 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  $\geq 20$  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 16-143. 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	<i>empty entry reserved for future use</i>	0x0000_000F
5	0	0	0	000_000	00	1111	0x0000_0000	<i>empty entry reserved for future use</i>	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

### 16.6.4.3 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 as follows:

1. Decide on a TxBD queue
2. Transmit exactly one frame from that queue
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.

### 16.6.4.3.1 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

```

---

### 16.6.4.3.2 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;
    endloop
endloop

```

---

```

        if ring_empty(ring) then
            credit[ring] = 0;
        endif
    endwhile
endloop

```

The algorithm checks registers TQUEUE[EN0–EN7] for `enabled()`, TSTAT[THLT0–THLT7] for `ring_empty()`, and TRxWT 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) \*  $WT_k / (\text{sum}(WT_i) + 6WT_0)$   
rate of queue[0] = (available bandwidth) \*  $7 * WT_0 / (\text{sum}(WT_i) + 6WT_0)$   
where  $i = 0$  to  $7$

## 16.6.5 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 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.

### 16.6.5.1 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:

$$\text{FreeBDsRequired} = \frac{\text{MaxFrameSize}}{\text{MinFrameSize} + \text{IFG}} + \frac{\text{MaxFrameSize}}{\text{RxBufferSize}} + \text{LinkDelay}$$

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 (96-bit 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 four BDs 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<sub>n</sub>) 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 RFBPTR<sub>n</sub> < RBPTR<sub>n</sub>

$$\text{FreeBDs} = \text{RQPRM}_n[\text{LEN}] - \text{RBPTR}_n + \text{RFBPTR}_n$$

When RFBPTR<sub>n</sub> > RBPTR<sub>n</sub>

$$\text{FreeBDs} = \text{RFBPTR}_n - \text{RBPTR}_n$$

When RBPTR<sub>n</sub> = RFBPTR<sub>n</sub> the number of free BDs in the ring is either one (since RFBPTR<sub>n</sub> points to a free BD) or equal to the ring length. Since the BD pointed to by RBPTR<sub>n</sub> 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<sub>n</sub> was updated and thus initializes both RBPTR<sub>n</sub> and RFBPTR<sub>n</sub>, the ring is deemed empty.

If RFBPTR<sub>n</sub> is updated by a software write and matches RBPTR<sub>n</sub>, the ring is deemed empty.

If the hardware updates RBPTR<sub>n</sub> and the result matches RFBPTR<sub>n</sub>, 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<sub>n</sub>, 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<sub>n</sub> to the next address (thereby equalling RBPTR<sub>n</sub>), 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<sub>n</sub>

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 [Section 16.6.2.8, “Flow Control”](#)) 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.

## 16.6.5.2 Software Use of Hardware-Initiated Back Pressure

This section discusses the initialization and operation of hardware-initiated back pressure.

### 16.6.5.2.1 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  $RQPRM_n[FBTHR]$ . 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),  $RFBPTR_n$  advances and the free BD count reflects the correct number of available free BDs.

### 16.6.5.2.2 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  $RFBPTR_n$  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.



## 16.6.6 Hardware Assist for IEEE Std. 1588 Compliant Timestamping

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 10  $\mu$ sec accuracy needs 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.

### 16.6.6.1 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 filter 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 filter 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

### 16.6.6.2 Timer Logic Overview

The 1588 timer module can be partitioned into four different sub-modules as shown in Figure 16-126.

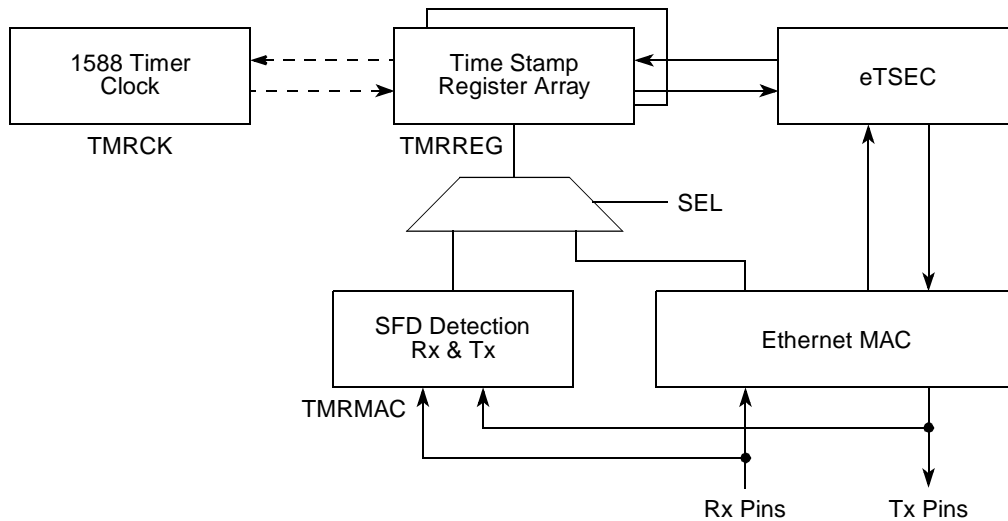


Figure 16-126. 1588 Timer Design Partition

### 16.6.6.3 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.

#### 16.6.6.3.1 Timestamp Point

The required timestamp point, as specified in the IEEE 1588 Specification Sep-2004 (IEC 61588 First Edition), is shown in Figure 16-127. From this, it is clear that the end of the SFD is the critical point in the MII data stream.

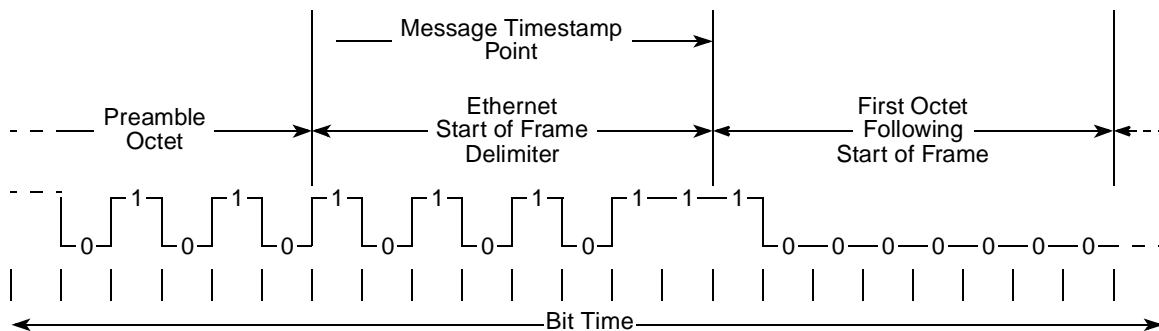


Figure 16-127. Ethernet Sampling Points for 1588

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) 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). 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.

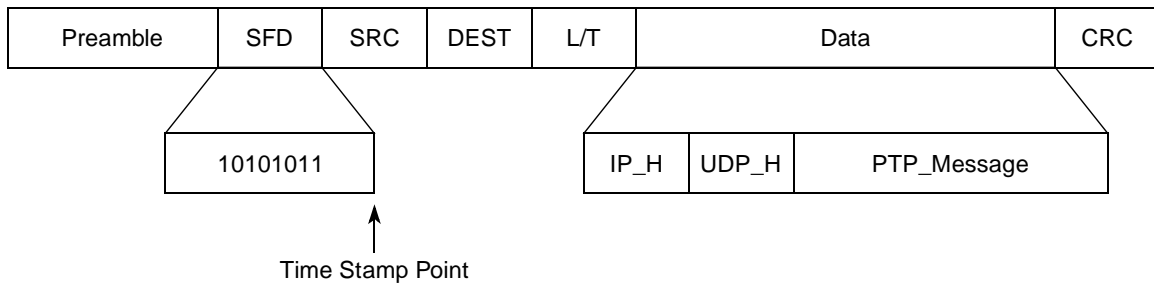
#### 16.6.6.4 PTP Packet Parsing

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 16-144.

**Table 16-144. PTP Payload Special Fields**

Layer	Octet (Offset from the SFD)	Field	Value	eTSEC filer PID	Comments
Ethernet	12-13	Length/Packet	0x0800	ETY-RQPFR[P ID=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[P ID=1011]	UDP
IP header	26-29	Source IP Address IANA defines 4 multicast address for the PTP packet		SIA-RQPFR[PI D=1101]	
IP header	30-33	Destination IP Address IANA defines 4 multicast address for the PTP packet	224.0.1.129 224.0.1.130 224.0.1.131 224.0.1.132	DIA-RQPFR[PI D=1100]	DefaultPTPdomain AlternatePTPdomain1 AlternatePTPdomain2 AlternatePTPdomain3
UDP header	34-35	Source port number		SPT-RQPFR[P ID=1011]	
UDP header	36-37	Destination port number	319 320	DPT-RQPFR[P ID=1011]	EventPort GeneralPort
UDP data	74	Control	0x0 0x1 0x2 0x3 0x4	RBIFX- choose an arbitrary extraction byte	Sync Delay_req Follow_up Delay_resp Management

A representation of the PTP packet is shown in [Figure 16-128](#).



**Figure 16-128. PTP Packet Format**

#### 16.6.6.4.1 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 using the filer to queue all PTP packets to one or more predefined virtual queues. See [Section 16.6.4.2.1, “Filing Rules”](#) for further information.

#### 16.6.6.5 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, that is, located in contiguous memory locations, as depicted in [Figure 16-129](#).

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 16-145](#). When time-stamps are inserted into the TxPAL, the TMR\_TXTSn\_H/L and TMR\_TXTSn\_ID registers still function normally.

##### 16.6.6.5.1 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\_TXTSn\_H/L and TMR\_TXTSn\_ID registers still function normally. Therefore, the 1588 interrupt can be triggered using the TMR\_PEVENT register bits TXP1, and TXP2.

**Table 16-145. 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.
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 16-129 depicts the buffer format requirements for time-stamp insertion on transmit packets.

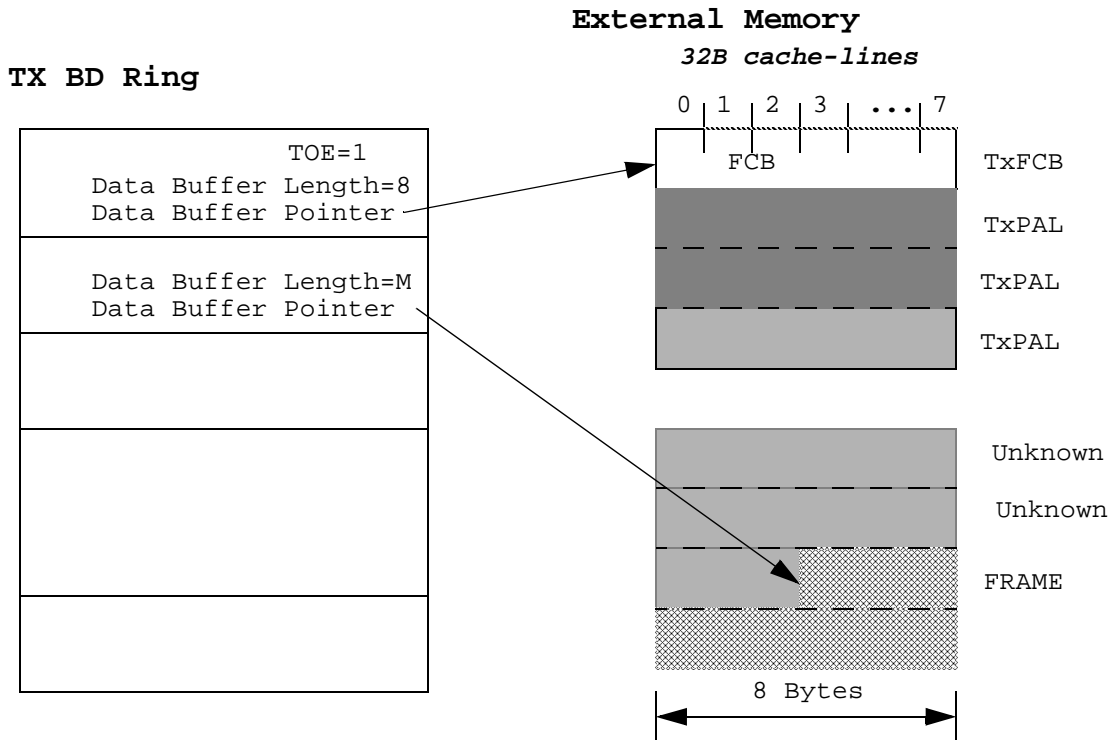


Figure 16-129. Buffer Format for Transmit Time-Stamp Insertion

### 16.6.6.5.2 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.

### 16.6.6.6 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 16-130.

	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 16-130. Transmit Frame Control Block

The contents of the Tx FCB are defined in [Table 16-146](#).

**Table 16-146. 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. 0 Ignore VLCTL field. 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. 0 Ignore layer 3 and higher headers. 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. 0 IP header version is 4. 1 IP header version is 6.
	3	TUP	Layer 4 header is a TCP or UDP header. 0 Do not process any layer 4 header. 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. 0 Layer 4 protocol is either TCP (if TUP = 1) or undefined. 1 Layer 4 protocol is UDP if TUP = 1.
0–1	5	CIP	Checksum IP header enable. 0 Do not generate an IP header checksum. 1 Generate an IPv4 header checksum.
	6	CTU	Checksum TCP or UDP header enable. 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. 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. 0 Calculate TCP or UDP pseudo-header checksum as normal, assuming that the IP header has no options. 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. 0 Do not attempt to capture transmission event time 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.

**Table 16-146. Tx Frame Control Block Description (continued)**

Bytes	Bits	Name	Description
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/ 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.

## 16.6.7 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.

### 16.6.7.1 Data Buffer Descriptors

Data buffers are used in the transmission and reception of Ethernet frames (see [Figure 16-131](#)). 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 16-132](#)), 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.



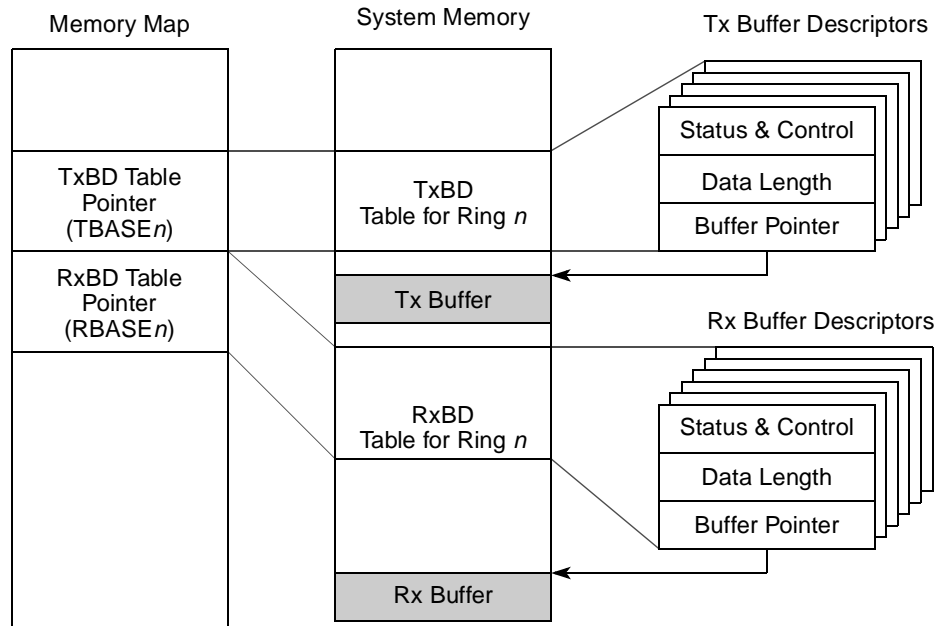


Figure 16-131. Example of eTSEC Memory Structure for BDs

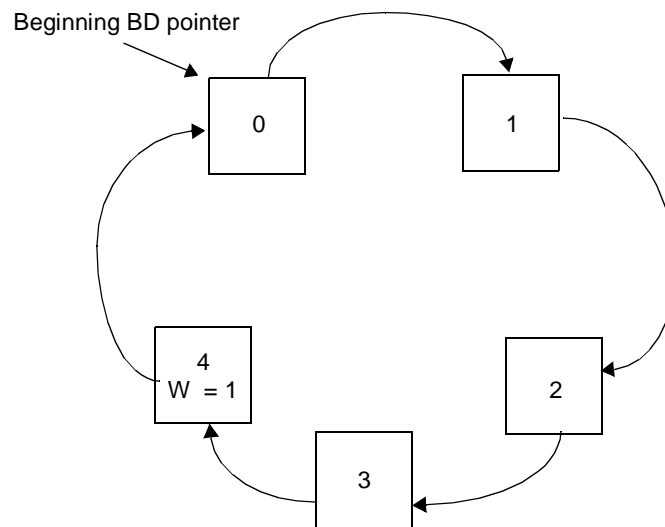


Figure 16-132. Buffer Descriptor Ring

### 16.6.7.2 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.

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 16-133 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 16-133. Transmit Buffer Descriptor**

The TxBD definition is interpreted by eTSEC hardware as if TxBDs mapped to C data structures in the manner shown in following code snippet.

```
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;
```

The TxBD fields are detailed in [Table 16-147](#).

**Table 16-147. Transmit Data Buffer Descriptor (TxBD) Field Descriptions**

Offset	Bits	Name	Description
0–1	0	R	Ready, written by eTSEC and user. 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. 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. 0 Do not add padding to short frames. 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. 0 The next buffer descriptor is found in the consecutive location. 1 The next buffer descriptor is found at the location defined in TBASE.
	3	I	Interrupt. Written by user. 0 No interrupt is generated after this buffer is serviced. 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. 0 The buffer is not the last in the transmit frame. 1 The buffer is the last in the transmit frame.

Table 16-147. 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. 0 End transmission immediately after the last data byte with no hardware generated CRC appended, unless TxBD[PAD/CRC] is set. 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. 0 This frame does not contain Ethernet preamble bytes for transmission. 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) 0 This frame was not deferred. 1 This frame did not have a collision before it was sent but it was sent late because of deferring
	7	—	Reserved	
	8	HFE	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. 0 Truncate transmit frame if its length is greater than the MAC's maximum frame length. 1 Allow large frames to be transmitted without truncation.
			LC	Late collision. Written by the eTSEC. 0 No late collision. 1 A collision occurred after 64 bytes are sent. The eTSEC terminates the transmission and updates LC.
	9	CF	CF	Control Frame. Written by user. Valid only if set in the first BD of a frame. 0 Regular frame; transmission is deferred when eTSEC is in PAUSE. 1 Control frame; transmission starts even if eTSEC is in PAUSE.
			RL	Retransmission Limit. Written by the eTSEC. 0 Transmission before maximum retry limit is hit. 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	RC	Retry Count. Written by the eTSEC. 0 The frame is sent correctly the first time. 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 16-147. Transmit Data Buffer Descriptor (TxBD) Field Descriptions (continued)**

Offset	Bits	Name	Description
0–1	14	UN	Underrun. Written by the eTSEC. 0 No underrun encountered (data was retrieved from external memory in time to send a complete frame). 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. 0 No TCP/IP off-load acceleration is applied to the frame prior to transmission. 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.

### 16.6.7.3 Receive Buffer Descriptors (RxB D)

In the RxB D 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 RxB D 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 16.5.3.5.5, “Maximum Frame Length Register \(MAXFRM\),”](#) for information on setting the size of the buffer ring.

[Figure 16-134](#) defines the RxB D.

	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 16-134. 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 16-135](#).

```
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 16-135. Mapping of RxBDs to a C Data Structure**

[Table 16-148](#) describes the fields of the RxBD.

**Table 16-148. 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). 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. 1 The data buffer associated with this BD is empty, or reception is currently in progress.
	1	RO1	Receive software ownership bit. 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. 0 The next buffer descriptor is found in the consecutive location. 1 The next buffer descriptor is found at the location defined in RBASE.
	3	I	Interrupt, written by user. 0 No interrupt is generated after this buffer is serviced. 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. 0 The buffer is not the last in a frame. 1 The buffer is the last in a frame.
	5	F	First in frame, written by the eTSEC. 0 The buffer is not the first in a frame. 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. 0 The frame was received because of an address recognition hit. 1 The frame was received because of promiscuous mode.

Table 16-148. 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). 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). 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 64 bytes 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). 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). 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). 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.
2–3	0–15	Data Length	Data length, written by the eTSEC. 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. 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.

## 16.7 Initialization/Application Information

This section discusses the following:

- [Section 16.7.1, “Interface Mode Configuration”](#)
- [Section 16.7.2, “MAC: Half-Duplex Collision on FCS of Short Frame”](#)

## 16.7.1 Interface Mode Configuration

This section describes how to configure the eTSEC in different supported interface modes. These include the following:

- MII
- RGMII

The pinout, the data registers that must be initialized, as well as speed selection options are described.

### 16.7.1.1 MII Interface Mode

Table 16-149 describes the signal configurations required for MII interface mode.

**Table 16-149. MII Interface Mode Signal Configuration**

eTSEC Signals			MII Interface		
			Frequency [MHz] 125		
			Voltage [V] 3.3/2.5		
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
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
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
<b>Sum</b>		17	<b>Sum</b>		16



Table 16-150 describes the shared signals of the MII interface.

**Table 16-150. 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
<b>Sum</b>		<b>2</b>	<b>Sum</b>		<b>2</b>	

Table 16-151 describes the register initializations required to configure the eTSEC in MII mode.

**Table 16-151. MII Mode Register Initialization Steps**

Set Soft_Reset, MACCFG1[1000_0000_0000_0000_0000_0000_0000_0000]
Clear Soft_Reset, MACCFG1[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize MACCFG2, for MII, half duplex operation. Set I/F Mode bit, MACCFG2[0000_0000_0000_0000_0111_0001_0000_0100] (This example has Full Duplex = 0, Preamble count = 7, PAD/CRC append = 1)
Initialize ECNTRL, ECNTRL[0000_0000_0000_0000_0001_0000_0000_0000] (This example has Statistics Enable = 1)
Initialize MAC Station Address, MACSTNADDR2[0110_0000_0000_0010_0000_0000_0000_0000] Set station address to 02_60_8C_87_65_43, for example.
Initialize MAC Station Address, MACSTNADDR1[0100_0011_0110_0101_1000_0111_1000_1100] Set station address to 02_60_8C_87_65_43, for example.
Reset the management interface. MIIMCFG[1000_0000_0000_0000_0000_0000_0000_0111]
Setup the MII Mgmt clock speed, MIIMCFG[0000_0000_0000_0000_0000_0000_0000_0101] 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, MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000] 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). MIIMADD[0000_0000_0000_0000_0000_0000_0001_1100]
Perform an MII Mgmt write cycle to the external PHY Writing to MII Mgmt Control with 16-bit data intended for the external PHY register, MIIMCON[0000_0000_0000_0000_0000_0000_0000_0100]
Check to see if MII Mgmt write is complete Read MII Mgmt Indicator register and check for Busy = 0, MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000] This indicates that the write cycle was completed.

**Table 16-151. MII Mode Register Initialization Steps (continued)**

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. MIIMADD[0000_0000_0000_0000_0000_0000_0001_0111]
Perform an MII Mgmt write cycle to the external PHY. Write to MII Mgmt Control with 16-bit data intended for the external PHY register, MIIMCON[0000_0000_0000_0000_0000_0000_0000_0000]
Check to see if MII Mgmt write is complete. Read MII Mgmt Indicator register and check for Busy = 0, MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000] 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. MIIMADD[0000_0000_0000_0000_0000_0000_0000_0000]
Perform an MII Mgmt write cycle to the external PHY. Write to MII Mgmt Control with 16-bit data intended for the external PHY register, MIIMCON[0000_0000_0000_0000_00uu_00uu_0u00_0000] where u is user defined based on desired configuration.
Check to see if MII Mgmt write is complete Read MII Mgmt Indicator register and check for Busy = 0, MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000] 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. Set up the MII Mgmt for a read cycle to PHY MII Mgmt register (write the PHY address and Register address), MIIMADD[0000_0000_0000_0000_0000_0000_0000_0001] 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. Clear MIIMCOM[Read Cycle]. Set MIIMCOM[Read Cycle]. (Uses the PHY address (0) and Register address (1) placed in MIIMADD register), When MIIMIND[BUSY]=0, read the MIIMSTAT register and check bit 10 (AN Done and Link is up) MIIMSTAT ---> [0000_0000_0000_0000_0000_0000_0010_0100] 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, IEVENT[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize IMASK (Optional) IMASK[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize MACnADDR1/2 (Optional) MACnADDR1/2[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize GADDRn (Optional) GADDRn[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize RCTRL (Optional) RCTRL[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize DMACTRL (Optional) DMACTRL[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize (Empty) Transmit Descriptor ring and fill buffers with Data Initialize TBASE0–TBASE7, TBASE0–TBASE7[LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_L000]

**Table 16-151. MII Mode Register Initialization Steps (continued)**

Initialize (Empty) Receive Descriptor ring and fill with empty buffers Initialize RBASE0–RBASE7, RBASE0–RBASE7[LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_L000]
Enable Transmit Queues Initialize TQUEUE
Enable Receive Queues Initialize RQUEUE
Enable Rx and Tx, MACCFG1[0000_0000_0000_0000_0000_0000_0000_0101]

### 16.7.1.2 RGMII Interface Mode

Table 16-152 shows the signals configurations required for RGMII interface mode.

**Table 16-152. 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
GTX_CLK125/TX_CLK	I	1	GTX_CLK125	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
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
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		
<b>Sum</b>		<b>17</b>	<b>Sum</b>		<b>13</b>

Table 16-153 describes the shared signals for the RGMII interface.

**Table 16-153. Shared RGMII Signals**

eTSEC Signals	I/O	No. of Signals	GMI 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
<b>Sum</b>		<b>2</b>	<b>Sum</b>		<b>2</b>	

Table 16-154 describes the register initializations required to configure the eTSEC in RGMII mode.

**Table 16-154. RGMII Mode Register Initialization Steps**

Set Soft_Reset, MACCFG1[1000_0000_0000_0000_0000_0000_0000_0000]
Clear Soft_Reset, MACCFG1[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize MACCFG2, MACCFG2[0000_0000_0000_0000_0111_0010_0000_0101] (I/F Mode = 2, Full Duplex = 1)
Initialize ECNTRL, ECNTRL[0000_0000_0000_0000_0001_0000_0000_0000] (This example has RGMII 10Mbps mode, Statistics Enable = 1)
Initialize MAC Station Address, MACSTNADDR2[0110_0000_0000_0010_0000_0000_0000_0000] to 02608C:876543, for example.
Initialize MAC Station Address, MACSTNADDR1[0100_0011_0110_0101_1000_0111_1000_1100] to 02608C:876543, for example.
Setup the MII Mgmt clock speed, MIIMCFG[0000_0000_0000_0000_0000_0000_0000_0101] 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, MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000] 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), MIIMADD[0000_0000_0000_0000_0001_0001_0000_0100] 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. Write to MII Mgmt Control with 16-bit data intended for the external PHY AN Advertisement register, MIIMCON[0000_0000_0000_0000_u0uu_uuuu_uuuu_uuuu] Where u must be selected by the user for proper system configuration.
Check to see if MII Mgmt write is complete. Read MII Mgmt Indicator register and check for Busy = 0, MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000] This indicates that the write cycle was completed.

**Table 16-154. RGMII Mode Register Initialization Steps (continued)**

<p>Set up the MII Mgmt for a write cycle to the external PHY Control register (write the PHY address and Register address), MIIMADD[0000_0000_0000_0000_0001_0001_0000_0000] The control register (CR) is at offset address 0x00 from the external PHY address. (in this case 0x11)</p>
<p>Perform an MII Mgmt write cycle to the external PHY. Write to MII Mgmt Control with 16-bit data intended for the external PHY Control register, MIIMCON[0000_0000_0000_0000_0001_0010_0000_0000] This enables the external PHY to restart Auto-Negotiations using the configuration set in the AN Advertisement register.</p>
<p>Check to see if MII Mgmt write is complete. Read MII Mgmt Indicator register and check for Busy = 0, MIIMIND ---&gt; [0000_0000_0000_0000_0000_0000_0000_0000] This indicates that the write cycle was completed.</p>
<p>Check to see if PHY has completed Auto-Negotiation. Set up the MII Mgmt for a read cycle to the PHY MII Mgmt register (write the PHY address and Register address), MIIMADD[0000_0000_0000_0000_0000_0010_0000_0001] The PHY Status register is at address 0x1 and in this case the PHY Address is 0x2.</p>
<p>Perform an MII Mgmt read cycle of Status Register. Clear MIIMCOM[Read Cycle] Set MIIMCOM[Read Cycle] (Uses the PHY address (2) and Register address (2) placed in MIIMADD register) When MIIMIND[BUSY]=0, read the MIIMSTAT register and check bit 10. (AN Done) MIIMSTAT ---&gt; [0000_0000_0000_0000_0000_0000_0010_0000] Other information about the link is also returned. (Extend Status, No pre, Remote Fault, An Ability, Link status, extend Ability)</p>
<p>Perform an MII Mgmt read cycle of AN Expansion Register. Setup MIIMADD[0000_0000_0000_0000_0001_0001_0000_0110] Clear MIIMCOM[Read Cycle] Set MIIMCOM[Read Cycle] (Uses the PHY address (0x11) and Register address (6) placed in MIIMADD register) When MIIMIND[BUSY]=0, read the MII Mgmt AN Expansion register and check bits 13 and 14. (NP Able and Page Rx'd) MII Mgmt AN Expansion ---&gt; [0000_0000_0000_0000_0000_0000_0000_0110]</p>
<p>Perform an MII Mgmt read cycle of AN Link Partner Base Page Ability Register. (Optional) Setup MIIMADD[0000_0000_0000_0000_0001_0001_0000_0101] Clear MIIMCOM[Read Cycle] Set MIIMCOM[Read Cycle] (Uses the PHY address (0x11) and Register address (5) placed in MIIMADD register) When MIIMIND[BUSY]=0, read the MII Mgmt AN Link Partner Base Page Ability register and check bits 9 and 10. (Half and Full Duplex) MII Mgmt AN Link Partner Base Page Ability ---&gt; [0000_0000_0000_0000_0000_0000_000x_1x10_0000]</p>
<p>Clear IEVENT register, IEVENT[0000_0000_0000_0000_0000_0000_0000_0000]</p>
<p>Initialize IMASK (Optional) IMASK[0000_0000_0000_0000_0000_0000_0000_0000]</p>
<p>Initialize MACnADDR1/2 (Optional) MACnADDR1/2[0000_0000_0000_0000_0000_0000_0000_0000]</p>
<p>Initialize GADDR<sub>n</sub> (Optional) GADDR<sub>n</sub>[0000_0000_0000_0000_0000_0000_0000_0000]</p>

**Table 16-154. RGMII Mode Register Initialization Steps (continued)**

Initialize RCTRL (Optional) RCTRL[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize DMACTRL (Optional) DMACTRL[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize (Empty) Transmit Descriptor ring and fill buffers with Data Initialize TBASE0–TBASE7, TBASE0–TBASE7[LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_L000]
Initialize (Empty) Receive Descriptor ring and fill with empty buffers Initialize RBASE0–RBASE7, RBASE0–RBASE7[LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_L000]
Enable Transmit Queues Initialize TQUEUE
Enable Receive Queues Initialize RQUEUE
Enable Rx and Tx, MACCFG1[0000_0000_0000_0000_0000_0000_0000_0101]

### 16.7.2 MAC: Half-Duplex Collision on FCS of Short Frame

Half-duplex collision on FCS of short frame may cause Tx lockup. In the half-duplex mode, if a collision occurs in the FCS bytes of a short (less than 64 bytes) frame, then the Ethernet MAC may lock up and stop transmitting data or control frames. Only a reset of the controller can restore proper operation once it is locked up.

The following are the workarounds:

- Option 1: Set MACCFG2[PAD/CRC] = 1 that pads all short Tx frames to 64 bytes.
- Option 2: Use software-generated CRC (MACCFG2[PAD/CRC] = 0, MACCFG2[CRC EN] = 0, and TxBD[TC] = 0).

# Chapter 17

## I<sup>2</sup>C Interface

This chapter describes the inter-IC (IIC or I<sup>2</sup>C) bus interface implemented on this device.

### 17.1 Introduction

The inter-IC (IIC or I<sup>2</sup>C) 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 17-1 shows a block diagram of an instance of I<sup>2</sup>C interface.

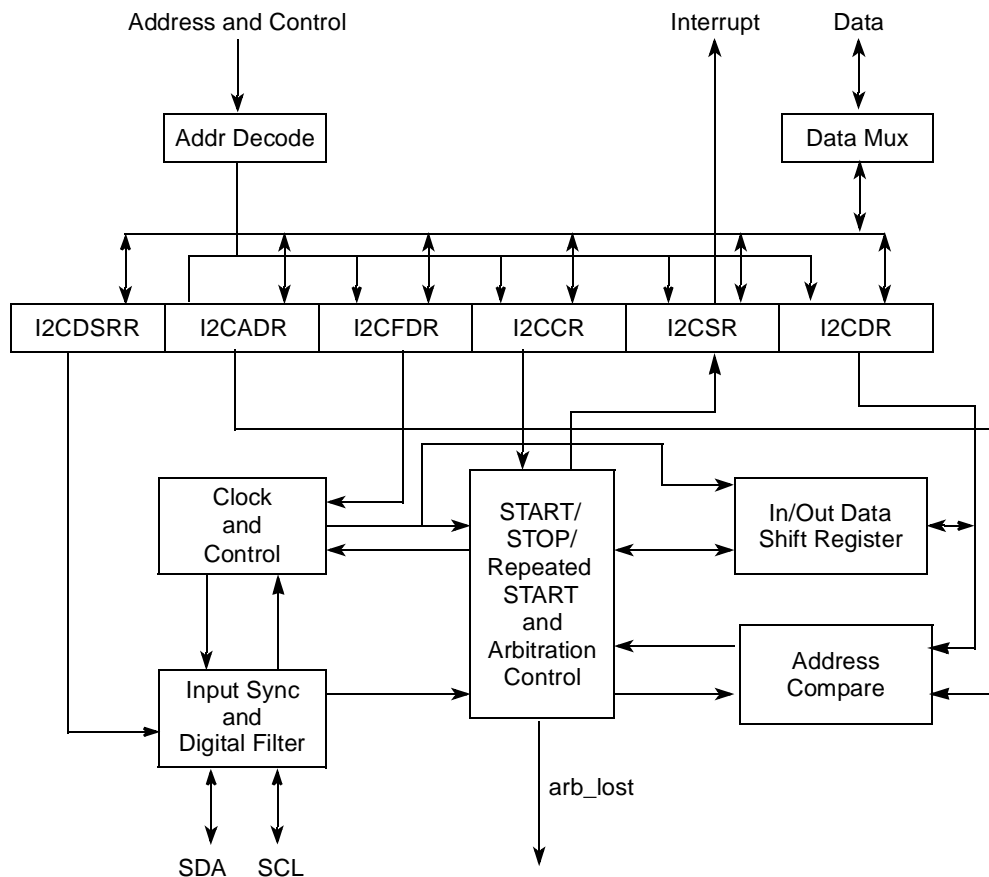


Figure 17-1. I<sup>2</sup>C Block Diagram

The two-wire I<sup>2</sup>C bus minimizes interconnections between devices. The synchronous, multiple-master I<sup>2</sup>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.

MPC8308 has two instances of I<sup>2</sup>C controllers. I<sup>2</sup>C controller 1 is used for boot sequencing and I<sup>2</sup>C controller 2 is used for data communication.

### 17.1.1 Features

The I<sup>2</sup>C interface includes the following features:

- Two-wire interface
- Multiple-master operational
- 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

### 17.1.2 Modes of Operation

The I<sup>2</sup>C unit on this device can operate in one of the following modes:

- Master mode.  
The I<sup>2</sup>C initiates a transfer, generates clock signals, and terminates a transfer. It cannot use its own slave address as a calling address. The I<sup>2</sup>C cannot be a master and a slave simultaneously.
- Slave mode.  
The I<sup>2</sup>C is addressed by an I<sup>2</sup>C master. The module must be enabled before a START condition from an I<sup>2</sup>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/ $\overline{W}$  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.  
I<sup>2</sup>C controller 1 supports boot sequencer mode. This mode can be used to initialize the configuration registers in the device after the I<sup>2</sup>C module is initialized. Boot sequencer mode is selected using the BOOTSEQ field in the reset configuration word high. Note that the hard-coded reset configuration word high value is boot sequencer mode disabled.
- Reset configuration load.  
In this mode, the I<sup>2</sup>C interface 1 loads the reset configuration words from an EEPROM at a specific calling address while the rest of the device is in the reset state ( $\overline{\text{HRESET}}$  asserted). Once the reset configuration words are latched inside the device, I<sup>2</sup>C is reset until  $\overline{\text{HRESET}}$  is negated. After



$\overline{\text{HRESET}}$  is negated, the device may be initialized using boot sequence mode according to the `BOOTSEQ` field in the reset configuration word. See [Section 17.4.5, “Boot Sequencer Mode.”](#)

Additionally, the following three I<sup>2</sup>C–specific states are defined for the I<sup>2</sup>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.

## 17.2 External Signal Descriptions

The following sections give an overview of signals and provide detailed signal descriptions.

### 17.2.1 Signal Overview

The I<sup>2</sup>C interface uses the SDA and SCL signals, described in [Table 17-1](#), 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 17-1. I<sup>2</sup>C Interface Signal Descriptions**

Signal Name	Idle State	I/O	State Meaning
Serial Clock (SCL1, SCL2)	High	I	When the I <sup>2</sup> 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 <sup>2</sup> C module drives SCL along with SDA when transmitting. As a slave, the I <sup>2</sup> C module drives SCL negates for data pacing.
Serial Data (SDA1, SDA2)	High	I	When the I <sup>2</sup> C module is idle or in a receiving mode, SDA defaults as an input. The unit receives data from other I <sup>2</sup> 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 <sup>2</sup> C module drives data on SDA synchronous to SCL.

### 17.2.2 Detailed Signal Descriptions

SDA and SCL, described in [Table 17-2](#), serve as a communication interconnect with other devices. All devices connected to these 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. For electrical characteristics, see *PowerQUICC II Pro MPC8308 Hardware Specification*.

**Table 17-2. I<sup>2</sup>C Interface Signals—Detailed Signal Descriptions**

Signal	I/O	Description
SCL	I/O	Serial clock. Performs as an input when the device is programmed as an I <sup>2</sup> C slave. SCL also performs as an output when the device is programmed as an I <sup>2</sup> C master.
	O	As outputs for the bidirectional serial clock, these signals operate as described below.
		<b>State Meaning</b>
	I	As inputs for the bi-directional serial clock, these signals operate as described below.
<b>State Meaning</b>		Asserted/Negated—The I <sup>2</sup> C unit uses this signal to synchronize incoming data on SDA. The bus is assumed to be busy when this signal is detected low.
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 <sup>2</sup> C master or a slave).
	O	As outputs for the bi-directional serial data, these signals operate as described below.
		<b>State Meaning</b>
	I	As inputs for the bi-directional serial data, these signals operate as described below.
<b>State Meaning</b>		Asserted/Negated—Used to receive data from other devices. The bus is assumed to be busy when SDA is detected low.

## 17.3 Memory Map/Register Definition

Table 17-3 lists the I<sup>2</sup>C-specific registers and their addresses.

**Table 17-3. I<sup>2</sup>C Memory Map**

Offset	I <sup>2</sup> C Register	Access	Reset	Section/Page
<b>I<sup>2</sup>C Controller 1—Block Base Address 0x0_3000</b> <b>I<sup>2</sup>C Controller 2—Block Base Address 0x0_3100</b>				
0x000	I2CADR—I <sup>2</sup> C address register	R/W	0x0000	<a href="#">17.3.1.1/17-5</a>
0x004	I2CFDR—I <sup>2</sup> C frequency divider register	R/W	0x0000	<a href="#">17.3.1.2/17-5</a>
0x008	I2CCR—I <sup>2</sup> C control register	R/W	0x0000	<a href="#">17.3.1.3/17-6</a>
0x00C	I2CSR—I <sup>2</sup> C status register	R/W	0x0081	<a href="#">17.3.1.4/17-8</a>
0x010	I2CDR—I <sup>2</sup> C data register	R/W	0x0000	<a href="#">17.3.1.5/17-9</a>
0x014	I2CDFSRR—I <sup>2</sup> C digital filter sampling rate register	R/W	0x0010	<a href="#">17.3.1.6/17-10</a>
0x01C–0x1FF	Reserved	—	—	—

## 17.3.1 Register Descriptions

This section describes the I<sup>2</sup>C registers in detail. Note that reserved bits should always be written with the value they return 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 does not apply to the I<sup>2</sup>C data register (I2CDR).

### 17.3.1.1 I<sup>2</sup>C Address Register (I2CADR)

Figure 17-2 shows the I2CADR register, which contains the address to which the I<sup>2</sup>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<sup>2</sup>C module is in master mode.

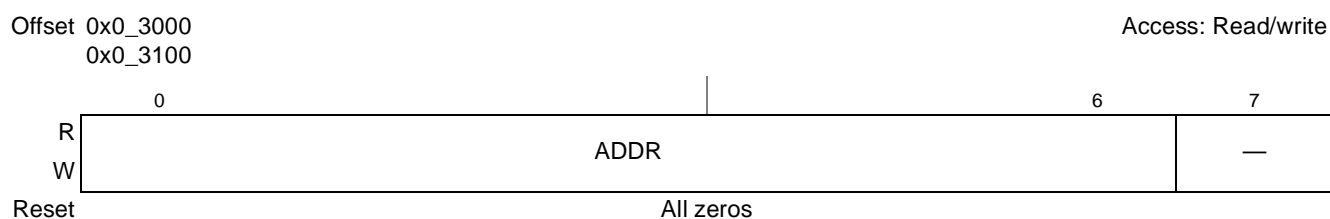


Figure 17-2. I<sup>2</sup>C Address Register (I2CADR)

Table 17-4 describes the bit settings of I2CADR.

Table 17-4. I2CADR Field Descriptions

Bits	Name	Description
0–6	ADDR	Slave address. Contains the specific slave address that is used by the I <sup>2</sup> C interface. Note that the default mode of the I <sup>2</sup> 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

### 17.3.1.2 I<sup>2</sup>C Frequency Divider Register (I2CFDR)

Figure 17-3 shows the bits of the I<sup>2</sup>C frequency divider register.

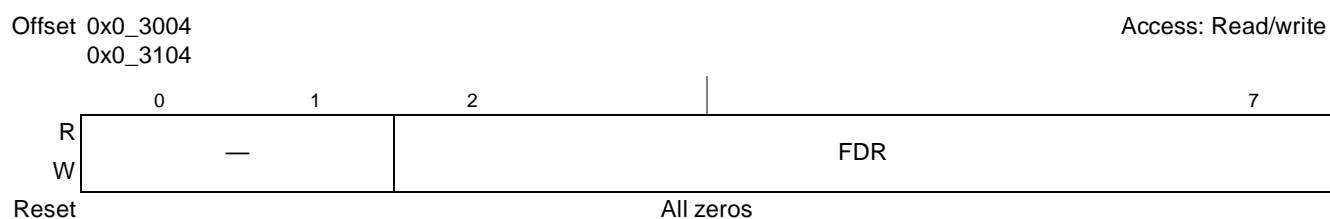


Figure 17-3. I<sup>2</sup>C Frequency Divider Register (I2CFDR)

Table 17-5 describes the bit settings of I2CFDR. It also maps I2CFDR[FDR] to the clock divider values. Although it describes the ratio between the I<sup>2</sup>C controller internal clock and SCL, the default ratio of I<sup>2</sup>C

controller clock and CSB is 1:1. Clock ratios for I<sup>2</sup>C1 as well as I<sup>2</sup>C2 are not programmable; they are always 1:1 with CSB. Consider this factor when selecting an FDR value.

**Table 17-5. I2C FDR Field Descriptions**

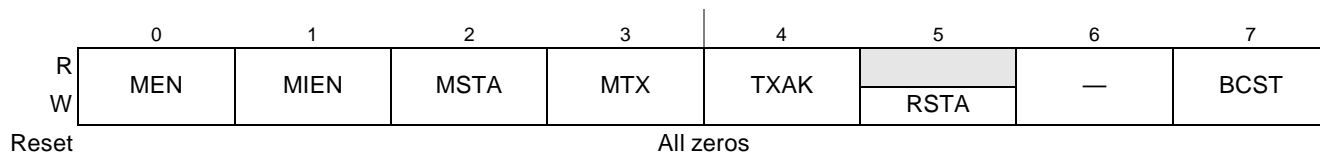
Bits	Name	Description																																																																																																																																										
0-1	—	Reserved, should be cleared																																																																																																																																										
2-7	FDR	<p>Frequency divider ratio. Used to prescale the clock for bit-rate selection. The serial bit clock frequency of SCL is equal to the I<sup>2</sup>C controller clock divided by the divider. The serial bit clock frequency divider selections are described as follows:</p> <table border="0"> <thead> <tr> <th><u>FDR</u></th> <th><u>Divider (Decimal)</u></th> <th><u>FDR</u></th> <th><u>Divider (Decimal)</u></th> <th><u>FDR</u></th> <th><u>Divider (Decimal)</u></th> </tr> </thead> <tbody> <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> </tbody> </table> <p><b>Note:</b> The value's shown in the table are applicable only for the default value of DFSRR. Refer to AN2919.</p>	<u>FDR</u>	<u>Divider (Decimal)</u>	<u>FDR</u>	<u>Divider (Decimal)</u>	<u>FDR</u>	<u>Divider (Decimal)</u>	0x00	384	0x16	12288	0x2B	1024	0x01	416	0x17	15360	0x2C	1280	0x02	480	0x18	18432	0x2D	1536	0x03	576	0x19	20480	0x2E	1792	0x04	640	0x1A	24576	0x2F	2048	0x05	704	0x1B	30720	0x30	2560	0x06	832	0x1C	36864	0x31	3072	0x07	1024	0x1D	40960	0x32	3584	0x08	1152	0x1E	49152	0x33	4096	0x09	1280	0x1F	61440	0x34	5120	0x0A	1536	0x20	256	0x35	6144	0x0B	1920	0x21	288	0x36	7168	0x0C	2304	0x22	320	0x37	8192	0x0D	2560	0x23	352	0x38	10240	0x0E	3072	0x24	384	0x39	12288	0x0F	3840	0x25	448	0x3A	14336	0x10	4608	0x26	512	0x3B	16384	0x11	5120	0x27	576	0x3C	20480	0x12	6144	0x28	640	0x3D	24576	0x13	7680	0x29	768	0x3E	28672	0x14	9216	0x2A	896	0x3F	32768	0x15	10240				
<u>FDR</u>	<u>Divider (Decimal)</u>	<u>FDR</u>	<u>Divider (Decimal)</u>	<u>FDR</u>	<u>Divider (Decimal)</u>																																																																																																																																							
0x00	384	0x16	12288	0x2B	1024																																																																																																																																							
0x01	416	0x17	15360	0x2C	1280																																																																																																																																							
0x02	480	0x18	18432	0x2D	1536																																																																																																																																							
0x03	576	0x19	20480	0x2E	1792																																																																																																																																							
0x04	640	0x1A	24576	0x2F	2048																																																																																																																																							
0x05	704	0x1B	30720	0x30	2560																																																																																																																																							
0x06	832	0x1C	36864	0x31	3072																																																																																																																																							
0x07	1024	0x1D	40960	0x32	3584																																																																																																																																							
0x08	1152	0x1E	49152	0x33	4096																																																																																																																																							
0x09	1280	0x1F	61440	0x34	5120																																																																																																																																							
0x0A	1536	0x20	256	0x35	6144																																																																																																																																							
0x0B	1920	0x21	288	0x36	7168																																																																																																																																							
0x0C	2304	0x22	320	0x37	8192																																																																																																																																							
0x0D	2560	0x23	352	0x38	10240																																																																																																																																							
0x0E	3072	0x24	384	0x39	12288																																																																																																																																							
0x0F	3840	0x25	448	0x3A	14336																																																																																																																																							
0x10	4608	0x26	512	0x3B	16384																																																																																																																																							
0x11	5120	0x27	576	0x3C	20480																																																																																																																																							
0x12	6144	0x28	640	0x3D	24576																																																																																																																																							
0x13	7680	0x29	768	0x3E	28672																																																																																																																																							
0x14	9216	0x2A	896	0x3F	32768																																																																																																																																							
0x15	10240																																																																																																																																											

### 17.3.1.3 I<sup>2</sup>C Control Register (I2CCR)

Figure 17-4 shows the I<sup>2</sup>C control register.

Offset 0x0\_3008  
0x0\_3108

Access: Mixed



**Figure 17-4. I<sup>2</sup>C Control Register (I2CCR)**

Table 17-6 describes the I2CCR bit settings.

**Table 17-6. I2CCR Field Descriptions**

Bits	Name	Description
0	MEN	Module enable. Controls the software reset of the I <sup>2</sup> C module. 0 The module is reset and disabled. The interface is held in reset, but the registers can still be accessed. 1 The I <sup>2</sup> C module is enabled. MEN must be set before any other control register bits have any effect. All I <sup>2</sup> C registers for slave receive or master START can be initialized before setting this bit.
1	MIEN	Module interrupt enable 0 Interrupts from the I <sup>2</sup> C module are disabled. This does not clear any pending interrupt conditions. 1 Interrupts from the I <sup>2</sup> C module are enabled. An interrupt occurs provided I2CSR[MIF] is also set.
2	MSTA	Master/slave mode START 0 On a transition to zero, a STOP condition is generated and the mode changes from master to slave. Cleared without generating a STOP condition when the master loses arbitration. 1 When MSTA changes from zero to one, a START condition is generated on the bus and master mode is selected.
3	MTX	Transmit/receive mode select. 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 will always high. MTX is cleared when the master loses arbitration. 0 Receive mode 1 Transmit mode
4	TXAK	Transfer acknowledge. Specifies the value driven onto the SDA line during acknowledge cycles for both master and slave receivers. The value of this bit applies only when the I <sup>2</sup> 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. 0 An acknowledge signal (low value on SDA) is sent out to the bus at the 9th clock bit after receiving one byte of data. 1 No acknowledge signal response (high value on SDA) is sent.
5	RSTA	Repeated START. Note that this bit is not readable, which means if a read is performed to RSTA, a zero value is returned. 0 No START condition is generated 1 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.
6	—	Reserved, should be cleared
7	BCST	Broadcast 0 Disables the broadcast accept capability 1 Enables the I <sup>2</sup> C to accept broadcast messages at address zero

### 17.3.1.4 I<sup>2</sup>C Status Register (I2CSR)

I2CSR is shown in Figure 17-5.

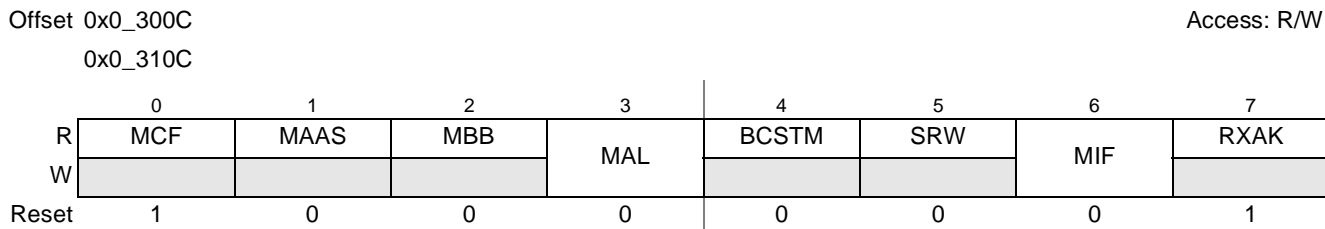


Figure 17-5. I<sup>2</sup>C Status Register (I2CSR)

Table 17-7 describes the bit settings of the I2CSR.

Table 17-7. 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. 0 Byte transfer in progress. MCF is cleared under the following conditions: <ul style="list-style-type: none"> <li>• When I2C DR is read in receive mode or when I2C DR is written in transmit mode.</li> <li>• After a start sequence is recognized by the I<sup>2</sup>C controller in slave mode.</li> </ul> 1 Byte transfer is completed
1	MAAS	Addressed as a slave. When the value in I2C ADR matches the calling address or when the calling address is the broadcast address and broadcast mode is enabled (I2CCR[BCST] is set), this bit is set. The processor is interrupted if I2CCR[MIE] is set. Next, the processor must check the SRW bit and set I2CCR[MTX] accordingly. Writing to the I2CCR automatically clears this bit. 0 Not addressed as a slave 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. 0 I <sup>2</sup> C bus is idle 1 I <sup>2</sup> 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. 0 Arbitration is not lost. Can only be cleared by software 1 Arbitration is lost
4	BCSTM	Broadcast match. Writing to the I2CCR automatically clears this bit. 0 There has not been a broadcast match. 1 The calling address matches with the broadcast address and broadcast mode is enabled. This is also set if this I <sup>2</sup> C drives an address of all 0s.
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. 0 Slave receive, master writing to slave 1 Slave transmit, master reading from slave. This bit is valid only when both of the following conditions are true: <ul style="list-style-type: none"> <li>• A complete transfer occurred and no other transfers have been initiated.</li> <li>• The I<sup>2</sup>C interface is configured as a slave and has an address match.</li> </ul> By checking SRW, the processor can select slave transmit/receive mode according to the command of the master.

Table 17-7. 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). 0 No interrupt is pending. Can be cleared only by software. 1 Interrupt is pending. MIF is set when one of the following events occurs: <ul style="list-style-type: none"> <li>• One byte of data is transferred (set at the falling edge of the 9th clock).</li> <li>• The value in I2CADDR matches with the calling address in slave-receive mode.</li> <li>• Arbitration is lost.</li> </ul>
7	RXAK	Received acknowledge. The value of SDA <sub>n</sub> 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. 0 Acknowledge received 1 No acknowledge received

### 17.3.1.5 I<sup>2</sup>C Data Register (I2CDR)

The I2C data register is shown in Figure 17-6.

Offset 0x0\_3010

Access: Read/write

0x0\_3110

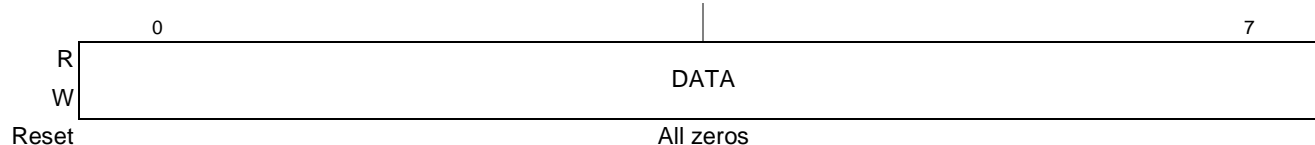


Figure 17-6. I<sup>2</sup>C Data Register (I2CDR)

Table 17-8 shows the bit descriptions for I2CDR.

Table 17-8. 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 <sup>2</sup> 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 <sup>2</sup> C module to receive the next byte of data on the I <sup>2</sup> C 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.

### 17.3.1.6 Digital Filter Sampling Rate Register (I2CDFSRR)

I2CDFSRR is shown in Figure 17-7.



Figure 17-7. I<sup>2</sup>C Digital Filter Sampling Rate Register (I2CDFSRR)

Table 17-9 shows the I2CDFSRR field descriptions.

Table 17-9. I2CDFSRR Field Descriptions

Bits	Name	Description
0–1	—	Reserved, should be cleared
2–7	DFSR	Digital filter sampling rate. To assist in filtering out signal noise, the sample rate is programmed. DFSR is used to prescale the frequency at which the digital filter takes samples from the I <sup>2</sup> C bus. The resulting sampling rate is calculated by dividing the platform frequency by the non-zero value of DFSR. If I2CDFSRR is cleared, the I <sup>2</sup> C bus sample points default to the reset divisor 0x10.

## 17.4 Functional Description

The I<sup>2</sup>C unit always performs as a slave receiver as a default, unless explicitly programmed to be a master or slave transmitter. If boot sequencer mode is selected, the I<sup>2</sup>C interface performs as a slave receiver after the boot sequence has completed.

### 17.4.1 Transaction Protocol

A standard I<sup>2</sup>C transfer consists of the following:

- START condition
- Slave target address transmission
- Data transfer
- STOP condition



Figure 17-8 shows the interaction of these four parts with the calling address, data byte, and new calling address components of the I<sup>2</sup>C protocol. The details of the protocol are described in the following subsections.

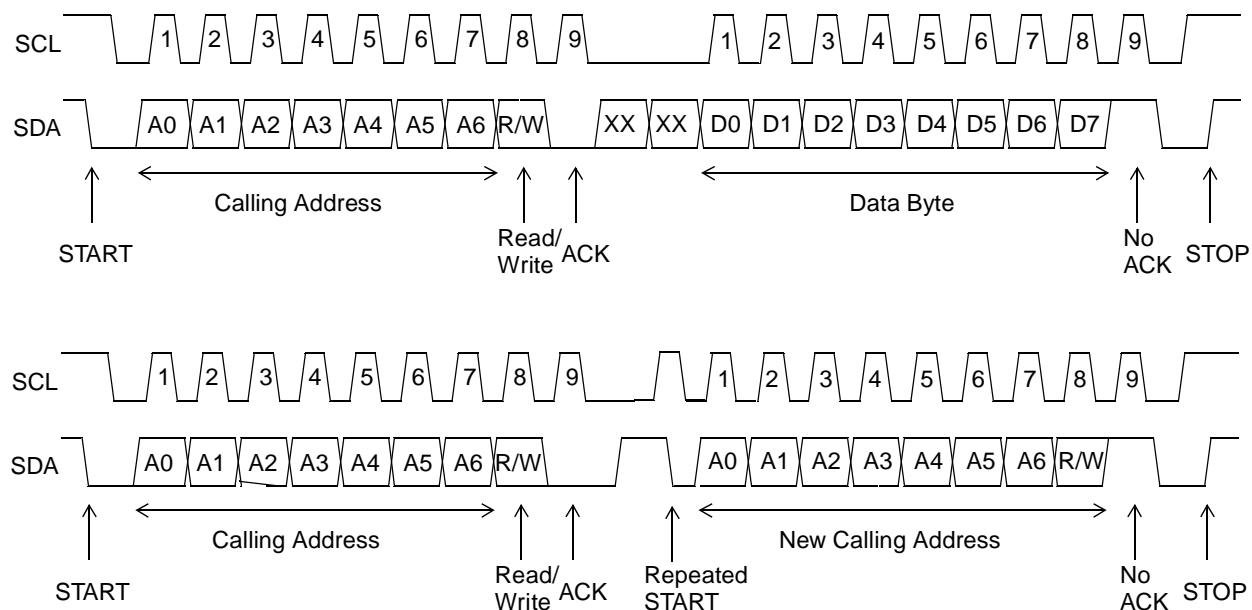


Figure 17-8. I<sup>2</sup>C Interface Transaction Protocol

### 17.4.1.1 START Condition

When the I<sup>2</sup>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 17-8, 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].

### 17.4.1.2 Slave Address Transmission

The first byte of data 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 transferred to the slave. Each slave in the system has a unique address. When the I<sup>2</sup>C module is operating as a master, it must not transmit an address that is the same as its slave address. An I<sup>2</sup>C device cannot be master and slave at the same time.

Only the slave with a calling address that matches the one transmitted by the master responds by returning an acknowledge bit (negating the SDA signal at the 9th clock), as shown in Figure 17-8. 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<sup>2</sup>C module responds to a general call (broadcast) command when I2CCR[BCST] is set. A broadcast address is always zero; however, the I<sup>2</sup>C module does not check the  $R/\overline{W}$  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, the 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 17-8](#). 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 SDA 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.

### 17.4.1.3 Repeated START Condition

[Figure 17-8](#) 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.

### 17.4.1.4 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 17-8](#). 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 17.4.1.3, “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.

### 17.4.1.5 Protocol Implementation Details

The following sections give details about how aspects of the protocol are implemented in the I<sup>2</sup>C module.

#### 17.4.1.5.1 Transaction Monitoring—Implementation Details

The different conditions of the I<sup>2</sup>C data transfers are monitored as follows (see [Figure 17-8](#)):

- 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.

#### 17.4.1.5.2 Control Transfer—Implementation Details

The I<sup>2</sup>C module contains logic that controls the output to the serial data (SDA) and serial clock (SCL) lines of the I<sup>2</sup>C. The SCL output is pulled low as determined by the internal clock generated in the clock module. The SDA output can change only at the midpoint of a low cycle of the SCL, unless it is performing a START, STOP, or repeated START condition. Otherwise, the SDA output is held constant.

SDA is negated when one or more of the following conditions are true:

- Master mode
  - Data bit (transmit)
  - ACK bit (receive)
  - START condition
  - STOP condition
  - Repeated START 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
  - Repeated START condition begin
  - Repeated START condition end
- Slave mode
  - Address cycle
  - Transmit cycle
  - ACK cycle

### 17.4.1.6 Address Compare—Implementation Details

The address compare block determines whether a slave has been properly addressed, either by its slave address or by the general broadcast address (which addresses all slaves). The following address comparisons are performed:

- Whether a broadcast message has been received, to update I2CSR
- Whether the module has been addressed as a slave, to update I2CSR and to generate an interrupt
- Whether the address transmitted by the current master matches the general broadcast address

## 17.4.2 Arbitration Procedure

The I<sup>2</sup>C interface is a true multiple-master bus. If two or more masters simultaneously try to control the bus, each master's clock synchronization procedure (including the I<sup>2</sup>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<sup>2</sup>C unit sets I2CSR[MAL] to indicate the loss of arbitration and, as a slave, services the transaction if it is directed to itself.

If the I<sup>2</sup>C module is enabled in the middle of an ongoing byte transfer, the interface behaves as follows:

- Slave mode—the I<sup>2</sup>C module ignores the current transfer on the bus and starts operating whenever a subsequent START condition is detected.
- Master mode—the I<sup>2</sup>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 causes in the current bus master to lose arbitration, after which bus operations return to normal.

### 17.4.2.1 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 repeated START 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 repeated START condition is attempted when the requesting device is not the bus owner
- Unexpected STOP condition detected

Note that the I<sup>2</sup>C module does not automatically retry a failed transfer attempt.

### 17.4.3 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.

### 17.4.4 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 repeated START condition
  - Transmit data
  - Receive data
- Slave mode
  - Transmit data
  - Receive data
  - Receive slave address after START or repeated START condition

#### 17.4.4.1 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 negates the SCL line. After a device has negated SCL, 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 SCL if another device is still within its low period. Therefore, 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 periods, SCL is released and asserted. Then there is no difference between the devices' clocks and the state of SCL, and all the devices begin counting their high periods. The first device to complete its high period negates SCL again.

#### 17.4.4.2 Input Synchronization and Digital Filter

The following sections describes synchronization of the input signals and the filtering of SCL and SDA in detail.

##### 17.4.4.2.1 Input Signal Synchronization

The input synchronization block synchronizes the input SCL and SDA signals to the system clock and detects transitions of these signals.

### 17.4.4.2.2 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 I2CDFSRR to control the filtered sampling rate.

### 17.4.4.3 Clock Stretching

Slaves can use the clock synchronization mechanism to slow down the transfer bit rate. After the master has driven SCL 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, the resulting SCL low period is extended.

## 17.4.5 Boot Sequencer Mode

Boot sequencer mode is selected at power-on reset by the BOOTSEQ field of the high-order reset configuration word. If boot sequencer mode is selected, the I<sup>2</sup>C module communicates with one or more EEPROMs through the I<sup>2</sup>C interface. EEPROMs can be programmed to initialize one or more configuration registers. Note that as described in [Section 4.3.2.2.2, “Boot Sequencer Configuration,”](#) the default value for BOOTSEQ is 0b00, which corresponds to the I<sup>2</sup>C boot sequencer being disabled at power-up.

Boot sequencer mode also supports an extension of the standard I<sup>2</sup>C interface that uses more address bits to allow for EEPROM devices that have more than 256 bytes. This extended addressing mode is selectable using a different encoding in the BOOTSEQ field of the high-order reset configuration word (see [Section 4.3.2.2.2, “Boot Sequencer Configuration.”](#)) In this mode, only one EEPROM device can be used and the maximum number of registers is limited by the size of the EEPROM.

If the standard I<sup>2</sup>C interface is used, the I<sup>2</sup>C module addresses the first EEPROM, and reads 256 bytes. Then it issues a repeated start and addresses the next EEPROM address. This sequence continues until the CONT bit is cleared. If the last register is not detected before wrapping back to the first address, an error condition is detected. In other words, if the CONT bit for not cleared on the final 7 bytes, an error condition is detected, causing the I<sup>2</sup>C controller to hang. The I<sup>2</sup>C module continues to read from the EEPROM as long as the continue (CONT) bit is set in the EEPROM. The CONT bit resides in the address/attributes field that is transferred from the EEPROM, as described in [Section 17.4.5.2, “EEPROM Calling Address.”](#) There should be no other I<sup>2</sup>C traffic when the boot sequencer is active.

### 17.4.5.1 Using the Boot Sequencer for Reset Configuration

The reset configuration word can be loaded by using the I<sup>2</sup>C boot sequencer. See [Section 4.3.2.2.2, “Boot Sequencer Configuration.”](#)

Note that this usage does not prevent using the I<sup>2</sup>C boot sequencer to initiate the device in the normal functional mode, after reset state has completed. However, an I<sup>2</sup>C serial EEPROM of extended addressing type must be used and the first two EEPROM data structures must contain dedicated reset information.

#### 17.4.5.2 EEPROM Calling Address

The EEPROM calling address is 0b101\_0000. The first EEPROM to be addressed must be programmed to respond to this address, or an error is generated. Any additional EEPROMs are addressed in sequential order.

### 17.4.5.3 EEPROM Data Format

The I<sup>2</sup>C module expects a particular format for data to be programmed in the EEPROM. Figure 17-9 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[12–13]			First Configuration Preload Command
ADDR[14–21]								
ADDR[22–29]								
DATA[0–7]								
DATA[8–15]								
DATA[16–23]								
DATA[24–31]								



ACS	BYTE_EN				1	ADDR[12–13]		Second Configuration Preload Command
ADDR[14–21]								
ADDR[22–29]								
DATA[0–7]								
DATA[8–15]								
DATA[16–23]								
DATA[24–31]								
.....								
ACS	BYTE_EN				1	ADDR[12–13]		Last Configuration Preload Command
ADDR[14–21]								
ADDR[22–29]								
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	
CRC[0–7]								
CRC[8–15]								
CRC[16–23]								
CRC[24–31]								

**Figure 17-9. EEPROM Contents**

- A preamble should be the first 3 bytes programmed into the EEPROM. It should have a value of 0xAA\_55AA. The I<sup>2</sup>C checks to ensure that this preamble is correctly detected before proceeding.
- Following the preamble, there should be a series of configuration registers (known as register preloads). Each configuration register should be programmed according to a particular format, as shown in [Figure 17-10](#).

0	1	2	3	4	5	6	7
ACS	BYTE_EN			CONT	ADDR[12:13]		
ADDR[14:21]							
ADDR[12:29]							
DATA[0:7]							
DATA[8:15]							
DATA[16:23]							
DATA[24:31]							

**Figure 17-10. EEPROM Data Format for One Register Preload Command**

- The first byte holds alternate configuration space (ACS), byte enables, and continue (CONT) attributes.
- The 2 least-significant bits of the address are derived from BYTE\_ENABLES[ADDRESS\_OFFSET]. Therefore, the address offset programmed into the EEPROM preload should be a word offset.
- The most significant 16 bits (assuming 36-bit addressing) of the address are prepended from either IMMRBAR or alternate configuration space.
- After the first 3 bytes, 4 bytes of data should hold the desired value of the configuration register, regardless of the size of transaction.

Byte enables should be asserted for any byte that will be written, 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 least-significant byte of data (data[24–31]).

By asserting ACS, an alternate configuration space address is prepended to the write request from the boot sequencer according to the value in the ALTGBAR register. This will allow for external memories to be configured. Otherwise, IMMRBAR is prepended to the EEPROM address.

If the CONT bit is cleared, the first 3 bytes, including ACS, the byte enables, and the address, should be cleared 0. Also, the data contains the final CRC. A CRC-32 algorithm is used to check the integrity of the data. The following polynomial is used:

$$1 + x^1 + x^2 + x^4 + x^5 + x^7 + x^8 + x^{10} + x^{11} + x^{12} + x^{16} + x^{22} + x^{23} + x^{26} + x^{32}$$

The CRC should cover all bytes stored in the EEPROM before 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).

#### 17.4.5.4 Boot Sequencer Done Indication

Dedicated hardware is not provided to indicate whether the boot sequencer operation completed successfully. It is recommended that one of the GPIO signals be used for that purpose. To do this, the last register preload programmed into the EEPROM should contain the address of the appropriate GPIO

register and data that causes the setting of the required GPIO signal. The GPIO signal may be used for an external device or for debug purposes.

## 17.5 Initialization/Application Information

This section describes some programming guidelines recommended for the I<sup>2</sup>C interface. Figure 17-11 is a recommended flowchart for I<sup>2</sup>C interrupt service routines.

A **sync** assembly instruction must be executed after each I<sup>2</sup>C register read/write access to guarantee that register accesses occur in order.

The I<sup>2</sup>C controller does not guarantee its recovery from all illegal I<sup>2</sup>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<sup>2</sup>C bus hangs. The recovery routine should also handle the case when the illegal I<sup>2</sup>C bus behavior causes the status bits returned after an interrupt to be inconsistent with what was expected.

### 17.5.1 Interrupt Service Routine Flowchart

Figure 17-11 shows an example algorithm for an I<sup>2</sup>C interrupt service routine. Deviation from the flowchart may result in unpredictable I<sup>2</sup>C bus behavior. However, in the slave receive mode (not shown), the interrupt service routine may need to set I2CCR[TXAK] when the next-to-last byte is to be accepted.

It is recommended that a **sync** instruction follow each I<sup>2</sup>C register read or write to guarantee that register accesses occur in order.

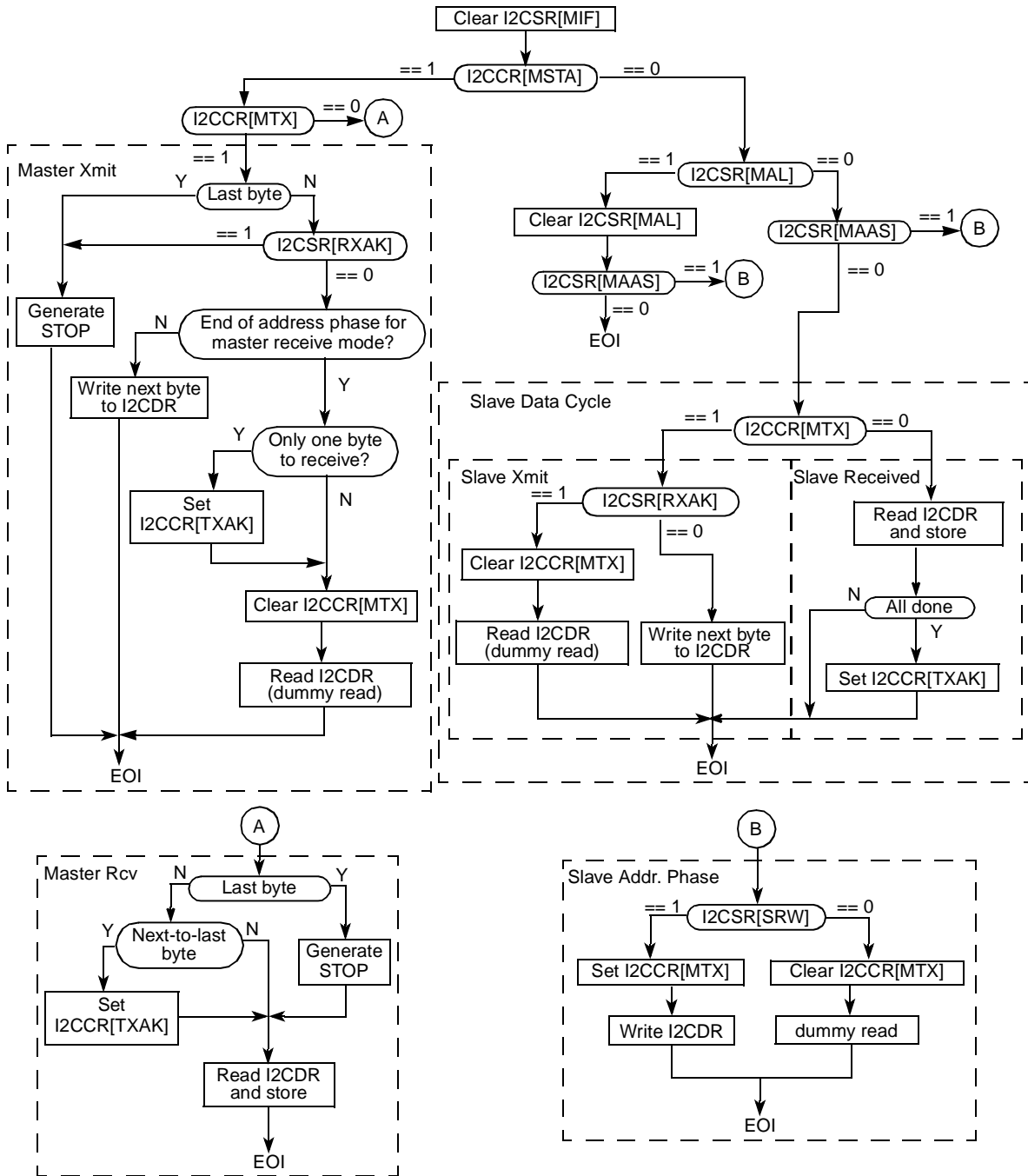


Figure 17-11. Example I<sup>2</sup>C Interrupt Service Routine Flowchart

## 17.5.2 Initialization Sequence

A hard reset initializes all of the I<sup>2</sup>C registers to their default states. The following initialization sequence initializes the I<sup>2</sup>C unit:

1. All I<sup>2</sup>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 CSB (platform) clock.
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<sup>2</sup>C interface.

## 17.5.3 Generation of START

After initialization, the following sequence can be used to generate START:

1. If the device is connected to a multimaster I<sup>2</sup>C system, 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] forms the slave calling address. I2CCR[MTX] indicates the direction of transfer (transmit/receive) required from the slave.

The scenario above assumes that the I<sup>2</sup>C interrupt bit (I2CSR[MIF]) is cleared. If MIF is set at any time, an I<sup>2</sup>C interrupt is generated (provided interrupt reporting is enabled with I2CCR[MIEN] = 1).

## 17.5.4 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<sup>2</sup>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 I2CDR in receive mode or write to I2CDR in transmit mode. Note that this causes I2CSR[MCF] to be cleared, as shown in [Figure 17-11](#).
3. 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 [Figure 17-11](#)).

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<sup>2</sup>C signals have time to settle. Thus, when polling I2CSR[MIF] (or any other I2CSR bits), software delays may be needed to give the I<sup>2</sup>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 [Figure 17-11](#)).

### 17.5.5 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 been transferred on the I<sup>2</sup>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.

### 17.5.6 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].

### 17.5.7 Generation of SCL When SDA is Negated

It is sometimes necessary to force the I<sup>2</sup>C module to become the I<sup>2</sup>C bus master out of reset and drive SCL<sub>n</sub> (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<sup>2</sup>C devices to be reset. Thus, SDA can be negated low by another I<sup>2</sup>C device while this I<sup>2</sup>C module is coming out of reset and will stay low indefinitely. The following procedure can be used to force this I<sup>2</sup>C module to generate SCL so that the device driving SDA can finish its transaction:

1. Disable the I<sup>2</sup>C module and set the master bit by setting I2CCR to 0x20.
2. Enable the I<sup>2</sup>C module by setting I2CCR to 0xA0.
3. Read I2CDR.
4. Return the I<sup>2</sup>C module to slave mode by setting I2CCR to 0x80.

### 17.5.8 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 $\overline{W}$  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 negates SCL between byte transfers. SCL is released when I2CDR is accessed in the required mode.

### 17.5.8.1 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 [Figure 17-11](#).

### 17.5.8.2 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 17.4.2.1, “Arbitration Control.”](#)





# Chapter 18

## DUART

This chapter describes the two (dual) universal asynchronous receiver/transmitters (UARTs) of the device. It describes the functional operation, the DUART initialization sequence, and the programming details for the DUART registers and features.

### 18.1 Overview

The DUART consists of two (dual) 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 system 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 18-1, each UART module consists of the following:

- Receive and transmit buffers
- 16-bit counter for baud rate generation
- Interrupt control logic

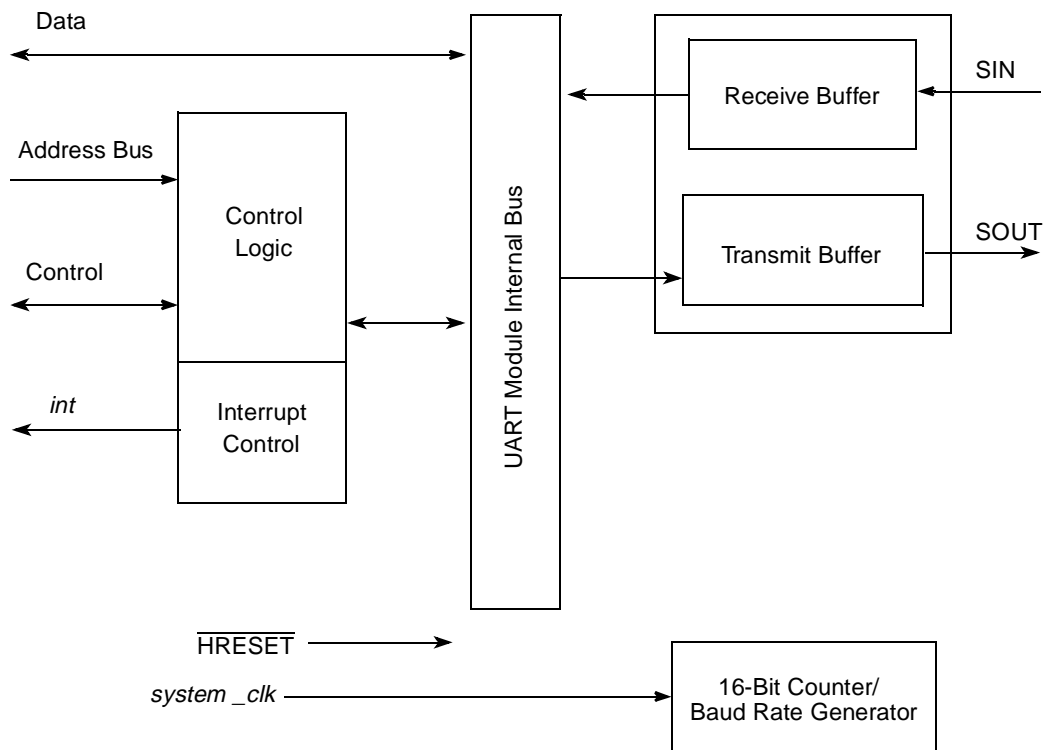


Figure 18-1. UART Block Diagram

## 18.1.1 Features

The DUART includes these features:

- Full-duplex operation
- Programming model compatible with the original PC16450 UART and the PC16550D (an improved version of the 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 system clock by 1 to  $(2^{16}-1)$  and generate a 16x clock for the transmitter and receiver engines
- 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

## 18.1.2 Modes of Operation

The communication channel provides a full-duplex asynchronous receiver and transmitter using an operating frequency derived from the system 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), and 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.

## 18.2 External Signal Descriptions

This section contains a signal overview and detailed signal descriptions.

### 18.2.1 Signal Overview

Table 18-1 summarizes the DUART signals. 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 18-1. DUART Signal Overview

Signal Name	I/O	Pins	Reset Value	State Meaning
UART_SIN[1:2]	I	2	1	Serial in data UART1 and UART2
UART_SOUT[1:2]	O	2	1	Serial out data UART1 and UART2

### 18.2.2 Detailed Signal Descriptions

The DUART signals are described in detail in Table 18-2.

Table 18-2. DUART Signals—Detailed Signal Descriptions

Signal	I/O	Description
UART_SIN[1:2]/DSP_UART_SIN	I	Serial data in. Data is received on the receivers of UART1, UART2, or DSP_UART through its respective serial data input signal, with the least significant bit received first.
		<b>State Meaning</b> Asserted/Negated—Represents the data being received on the UART interface.
		<b>Timing</b> Assertion/Negation—An internal logic sample signal, <i>rxcnt</i> , uses the frequency of the baud-rate generator to sample the data on SIN.
UART_SOUT[1:2]/DSP_UART_SOUT	O	Serial data out. The serial data output signals for the UART1, UART2, or DSP_UART 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.
		<b>State Meaning</b> Asserted/Negated—Represents the data transmitted on the respective UART interface.
		<b>Timing</b> Assertion/Negation—An internal logic sample signal, <i>rxcnt</i> , uses the frequency of the baud-rate generator to update and drive the data on SOUT.

## 18.3 Memory Map/Register Definition

There are two complete sets of DUART registers (one for UART1 and one for UART2). The two UARTs are identical, except that the registers for UART1 are located at offsets 0x0\_4500 (local) and the registers for UART2 are located at offsets 0x0\_4600 (local). Throughout this chapter, the registers are described by a singular acronym: for example, LCR represents the line control register for either UART1 or UART2.

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 18.3.1.7, “Line Control Registers (ULCR1 and ULCR2),” for more information on ULCR[DLAB].

All DUART registers are one byte wide; reads and writes to these registers must be byte-wide operations. [Table 18-3](#) provides a register summary with references to the section and page that contain detailed information about each register. Undefined byte address spaces within offset 0x4000–0x4FFF are reserved.

**Table 18-3. DUART Register Summary**

Offset	Register	Access	Reset	Section/Page
<b>UART 1—Block Base Address 0x0_4000 UART 2—Block Base Address 0x0_4100</b>				
0x0_4500	URBR—ULCR[DLAB] = 0 UART1 receiver buffer register	R	0x0000	<a href="#">18.3.1.1/18-5</a>
	UTHR—ULCR[DLAB] = 0 UART1 transmitter holding register	W	0x0000	<a href="#">18.3.1.2/18-5</a>
	UDLB—ULCR[DLAB] = 1 UART1 divisor least significant byte register	R/W	0x0000	<a href="#">18.3.1.3/18-6</a>
0x0_4501	UIER—ULCR[DLAB] = 0 UART1 interrupt enable register	R/W	0x0000	<a href="#">18.3.1.4/18-7</a>
	UDMB—ULCR[DLAB] = 1 UART1 divisor most significant byte register	R/W	0x0000	<a href="#">18.3.1.3/18-6</a>
0x0_4502	UIIR—ULCR[DLAB] = 0 UART1 interrupt ID register	R	0x0001	<a href="#">18.3.1.5/18-8</a>
	UFCR—ULCR[DLAB] = 0 UART1 FIFO control register	W	0x0000	<a href="#">18.3.1.6/18-9</a>
	UAFR—ULCR[DLAB] = 1 UART1 alternate function register	R/W	0x0000	<a href="#">18.3.1.11/18-14</a>
0x0_4503	ULCR—ULCR[DLAB] = x UART1 line control register	R/W	0x0000	<a href="#">18.3.1.7/18-10</a>
0x0_4504	UMCR—ULCR[DLAB] = x UART1 MODEM control register	R/W	0x0000	<a href="#">18.3.1.8/18-12</a>
0x0_4505	ULSR—ULCR[DLAB] = x UART1 line status register	R	0x0060	<a href="#">18.3.1.9/18-13</a>
0x0_4506	Reserved	—	—	—
0x0_4507	USCR—ULCR[DLAB] = x UART1 scratch register	R/W	0x0000	<a href="#">18.3.1.10/18-14</a>
0x0_4510	UDSR—ULCR[DLAB] = x UART1 DMA status register	R	0x0001	<a href="#">18.3.1.12/18-15</a>
0x0_4600	URBR—ULCR[DLAB] = 0 UART2 receiver buffer register	R	0x0000	<a href="#">18.3.1.1/18-5</a>
	UTHR—ULCR[DLAB] = 0 UART2 transmitter holding register	W	0x0000	<a href="#">18.3.1.2/18-5</a>
	UDLB—ULCR[DLAB] = 1 UART2 divisor least significant byte register	R/W	0x0000	<a href="#">18.3.1.3/18-6</a>
0x0_4601	UIER—ULCR[DLAB] = 0 UART2 interrupt enable register	R/W	0x0000	<a href="#">18.3.1.4/18-7</a>
	UDMB—ULCR[DLAB] = 1 UART2 divisor most significant byte register	R/W	0x0000	<a href="#">18.3.1.3/18-6</a>
0x0_4602	UIIR—ULCR[DLAB] = 0 UART2 interrupt ID register	R	0x0001	<a href="#">18.3.1.5/18-8</a>
	UFCR—ULCR[DLAB] = 0 UART2 FIFO control register	W	0x0000	<a href="#">18.3.1.6/18-9</a>
	UAFR—ULCR[DLAB] = 1 UART2 alternate function register	R/W	0x0000	<a href="#">18.3.1.11/18-14</a>
0x0_4603	ULCR—ULCR[DLAB] = x UART2 line control register	R/W	0x0000	<a href="#">18.3.1.7/18-10</a>
0x0_4604	UMCR—ULCR[DLAB] = x UART2 MODEM control register	R/W	0x0000	<a href="#">18.3.1.8/18-12</a>
0x0_4605	ULSR—ULCR[DLAB] = x UART2 line status register	R	0x0060	<a href="#">18.3.1.9/18-13</a>
0x0_4606	Reserved	—	—	—

Table 18-3. DUART Register Summary (continued)

Offset	Register	Access	Reset	Section/Page
0x0_4607	USCR—ULCR[DLAB] = x UART2 scratch register	R/W	0x0000	18.3.1.10/18-14
0x0_4610	UDSR—ULCR[DLAB] = x UART2 DMA status register	R	0x0001	18.3.1.12/18-15

### 18.3.1 Register Descriptions

The following sections describe the UART1 and UART2 registers.

#### 18.3.1.1 Receiver Buffer Registers (URBR1 and URBR2)

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 18.3.1.9, “Line Status Registers \(ULSR1 and ULSR2\).”](#) [Figure 18-2](#) shows the receiver buffer registers. Note that these registers have same offset as the UTHR<sub>s</sub>.

Offset: 0x0\_4500, 0x0\_4600

Access: User read-only

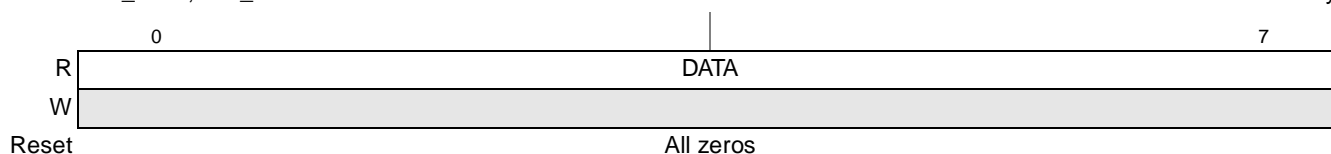


Figure 18-2. Receiver Buffer Registers (URBR1 and URBR2)

[Table 18-4](#) describes URBR.

Table 18-4. URBR Field Descriptions

Bits	Name	Description
0–7	DATA	Data received from the transmitter on the UART bus [read only]

#### 18.3.1.2 Transmitter Holding Registers (UTHR1 and UTHR2)

A write to these 8-bit registers causes the UART devices to transfer 5 to 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[TXRDY] indicates when the FIFO is full. Refer to [Table 18-20](#) and [Table 18-21](#).

Figure 18-3 shows the bits in the UTHR.

Offset: 0x0\_4500, 0x0\_4600

Access: User write-only

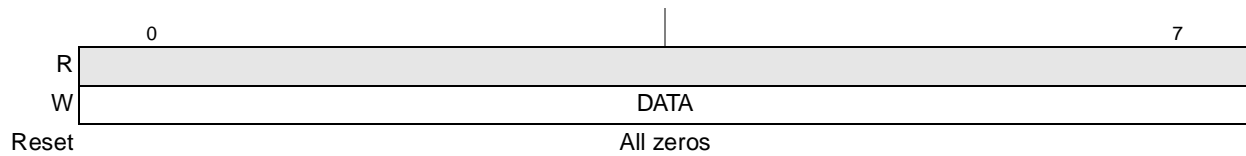


Figure 18-3. Transmitter Holding Registers (UTHR1 and UTHR2)

Table 18-5 describes the UTHR.

Table 18-5. UTHR Field Descriptions

Bits	Name	Description
0–7	DATA	Data that is written to UTHR [Write only]

### 18.3.1.3 Divisor Most and Least Significant Byte Registers (UDMB and UDLB)

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 18-8.

Figure 18-4 shows the bits in the UDMBs.

Offset: 0x0\_4501, 0x0\_4601

Access: User read/write

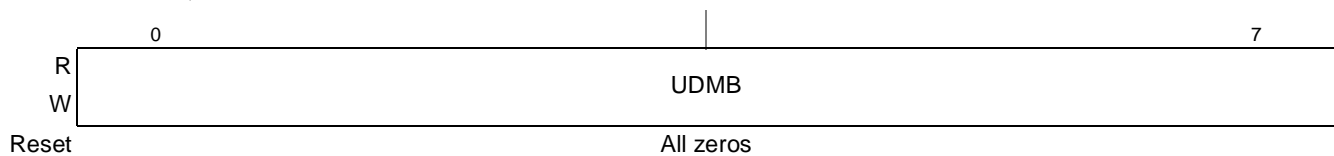


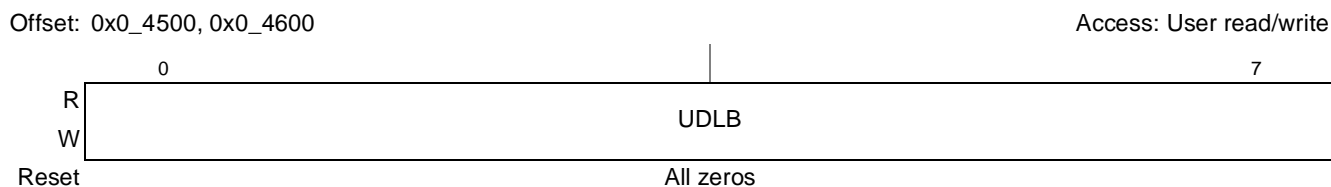
Figure 18-4. Divisor Most Significant Byte Registers (UDMB1 and UDMB2)

Table 18-6 describes the UDMB.

Table 18-6. UDMB Field Descriptions

Bits	Name	Description
0–7	UDMB	Divisor most significant byte

Figure 18-5 shows the bits in the UDLBs.



**Figure 18-5. Divisor Least Significant Byte Registers (UDLB1 and UDLB2)**

Table 18-7 describes the UDLB.

**Table 18-7. UDLB Field Descriptions**

Bits	Name	Description
0–7	UDLB	Divisor least significant byte. This is concatenated with UDMB.

Table 18-8 shows baud rate for a variety of input clock frequencies.

**Table 18-8. Baud Rate Examples**

Baud Rate (Decimal)	Divisor		Input Clock (System Clock) Frequency (MHz)	Percent Error (Decimal)
	Decimal	Hex		
9,600	866	362	133	0.013
19,200	433	1B1	133	0.013
38,400	216	D8	133	0.218
56,000	148	94	133	0.300
128,000	65	41	133	0.090
256,000	32	20	133	1.471

To get the percent error value, the following three steps are taken:

1. The input clock frequency (ICF) is divided by the actual frequency input (AFI) to get the correct divisor value (ICF/AFI, where AFI = baud rate × 16 × divisor).
2. The divisor value is subtracted from 1.
3. The result from the step two is multiplied by 100 to calculate the final percent error. The result is calculated in absolute value (no negative numbers).

These steps can be described with the following equation:

$$\text{Percent error value} = (1 - \text{AFI/ICF}) \times 100$$

### 18.3.1.4 Interrupt Enable Registers (UIER1 and UIER2)

The UIER gives the user the ability to mask specific UART interrupts to the programmable interrupt controller (PIC).

Figure 18-6 shows the bits in the UIER.

Offset: 0x0\_4501, 0x0\_4601

Access: User read/write

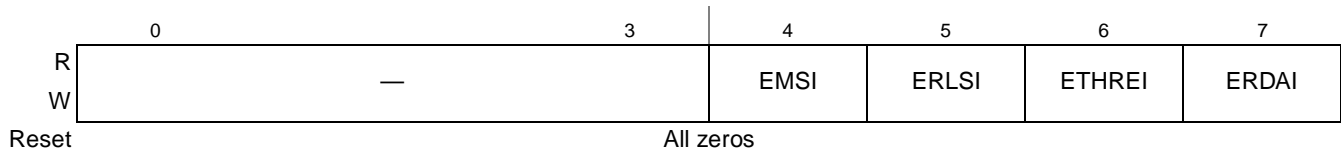


Figure 18-6. Interrupt Enable Registers (UIER1 and UIER2)

Table 18-9 describes the UIER fields.

Table 18-9. UIER Field Descriptions

Bits	Name	Description
0–3	—	Reserved
4	EMSI	Enable MODEM status interrupt 0 Mask interrupts caused by UMSR[DCTS] being set. 1 Enable and assert interrupts when UMSR[CTS] changes state.
5	ERLSI	Enable receiver line status interrupt 0 Mask interrupts when ULSR's overrun, parity error, framing error, or break interrupt bits are set. 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 0 Mask interrupt when ULSR[THRE] is set. 1 Enable and assert interrupts when ULSR[THRE] is set.
7	ERDAI	Enable received data available interrupt 0 Mask interrupt when new receive data is available or receive data time-out has occurred. 1 Enable and assert interrupts when a new data character is received from the external device and/or a time-out interrupt occurs in FIFO mode.

### 18.3.1.5 Interrupt ID Registers (UIR1 and UIR2)

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 as follows:

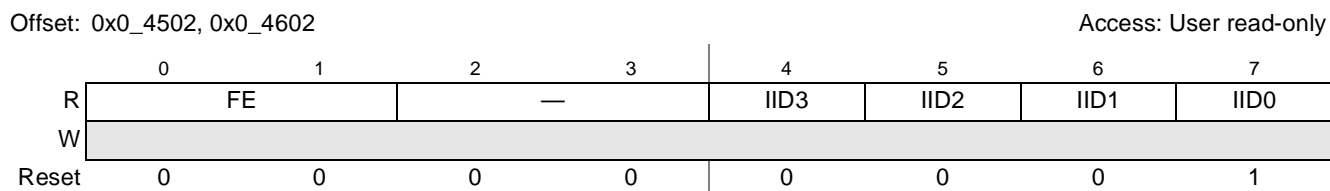
1. Receiver line status
2. Received data ready/character time-out
3. Transmitter holding register empty
4. MODEM status

See [Table 18-11](#) 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 18-7 shows the bits in the UIIR.



**Figure 18-7. Interrupt ID Registers (UIIR1 and UIIR2)**

Table 18-10 describes the fields of the UIIR.

**Table 18-10. 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 18-11. IID3 is set along with IID2 only when a time out interrupt is pending for FIFO mode.
5–6	IID2–IID1	Interrupt ID bits identify the highest priority pending interrupt as indicated in Table 18-11.
7	IID0	IID0 indicates when an interrupt is pending. 0 The UART has an active interrupt ready to be serviced. 1 No interrupt is pending.

The bits contained in the UIIR registers are described in Table 18-11.

**Table 18-11. UIIR IID Bits Summary**

IID3–IID0	Priority Level	Interrupt Type	Interrupt Description	How To Reset Interrupt
0001	—	—	—	—
0110	Highest	Receiver line status	Overrun error, parity error, framing error, or break interrupt	Reading the line status register
0100	Second	Received data available	Receiver data available or trigger level reached in FIFO mode.	Reading the receiver buffer register or if the number of bytes in the receiver FIFO drops below the trigger level.
1100	Second	Character time-out	No characters were removed from or input to the receiver FIFO during the last four character times and at least one character is in the receiver FIFO.	Reading the receiver buffer register
0010	Third	UTHR empty	Transmitter holding register is empty.	Reading UIIR or writing to UTHR

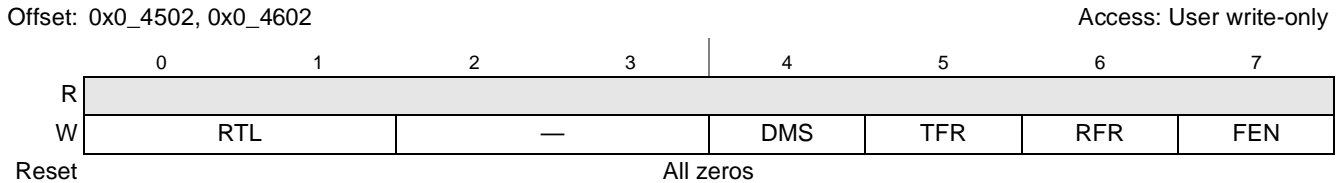
### 18.3.1.6 FIFO Control Registers (UFCR1 and UFCR2)

UFCR 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.

UFCR bits cannot be programmed unless FIFO enable bits are set. When changing from FIFO mode to 16450 mode (non-FIFO mode) and vice versa, data is automatically cleared from the FIFOs.

After all of 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.

Figure 18-8 shows the bits in the UFCRs.



**Figure 18-8. FIFO Control Registers (UFCR1 and UFCR2)**

Table 18-12 describes the fields of the UFCRs.

**Table 18-12. 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 RTL value. 00 1 byte 01 4 bytes 10 8 bytes 11 14 bytes
2–3	—	Reserved
4	DMS	DMA mode select. See <a href="#">Section 18.4.5.2, “DMA Mode Select”</a> 0 UDSR[RXRDY] and UDSR[TXRDY] bits are in mode 0. 1 UDSR[RXRDY] and UDSR[TXRDY] bits are in mode 1 if UFCR[FEN] = 1.
5	TFR	Transmitter FIFO reset 0 No action 1 Clears all bytes in the transmitter FIFO and resets the FIFO counter/pointer to 0
6	RFR	Receiver FIFO reset 0 No action 1 Clears all bytes in the receiver FIFO and resets the FIFO counter/pointer to 0
7	FEN	FIFO enable 0 FIFOs are disabled and cleared 1 Transmitter and receiver FIFOs are enabled.

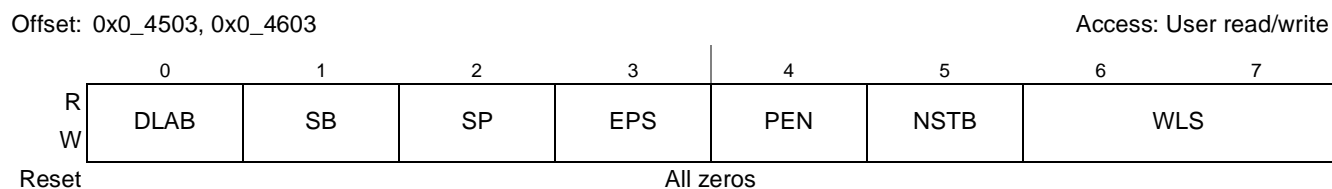
### 18.3.1.7 Line Control Registers (ULCR1 and ULCR2)

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 ULCR, the software should not rewrite the ULCR while valid transfers on the UART bus are active. The software should not rewrite the ULCR until the last STOP bit is received and no new characters are 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 18-14](#). ULCR[NSTB] defines the number of STOP bits to be sent at the end of the data transfer. The receiver checks only 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 transmitted or received as a serial character. The word length does not include START, parity, and STOP bits.

[Figure 18-9](#) shows the bits in the ULCRs.



**Figure 18-9. Line Control Register (ULCR1 and ULCR2)**

[Table 18-13](#) describes the ULCR fields.

**Table 18-13. ULCR Field Descriptions**

Bits	Name	Description
0	DLAB	Divisor latch access bit 0 Access to all registers except UDLB, UAFR, and UDMB. 1 Ability to access UDMB, UDLB, and UAFR.
1	SB	Set break 0 Send normal UTHR data onto the SOUT signal. 1 Force logic 0 to be on SOUT. Data in the UTHR is not affected.
2	SP	Stick parity 0 Stick parity is disabled. 1 If PEN = 1 and EPS = 1, space parity is selected; if PEN = 1 and EPS = 0, mark parity is selected.
3	EPS	Even parity select. See <a href="#">Table 18-14</a> . 0 If PEN = 1 and SP = 0 then odd parity is selected. 1 If PEN = 1 and SP = 0 then even parity is selected.
4	PEN	Parity enable 0 No parity generation and checking. 1 Generate parity bit as a transmitter, and check parity as a receiver.
5	NSTB	Number of STOP bits 0 One STOP bit is generated in the transmitted data. 1 When a 5-bit data length is selected, 1 1/2 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. 00 5 bits 01 6 bits 10 7 bits 11 8 bits

**Table 18-14. Parity Selection Using ULCR[PEN], ULCR[SP], and ULCR[EPS]**

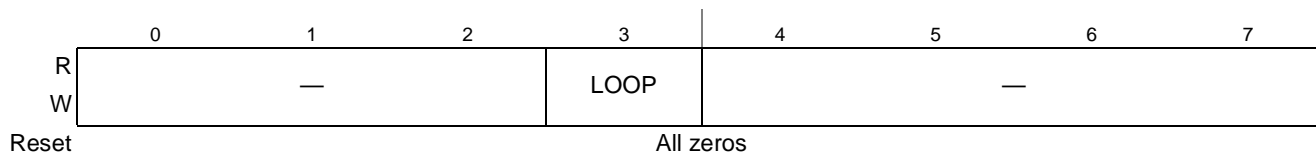
PEN	SP	EPS	Parity Selected
0	0	0	No parity
0	0	1	No parity
0	1	0	No parity
0	1	1	No parity
1	0	0	Odd parity
1	0	1	Even parity
1	1	0	Mark parity
1	1	1	Space parity

### 18.3.1.8 MODEM Control Registers (UMCR1 and UMCR2)

The UMCRs, shown in [Figure 18-10](#), control the interface with the external peripheral device on the UART bus.

Offset: 0x0\_4504, 0x0\_4604

Access: User read/write

**Figure 18-10. Modem Control Register (UMCR1 and UMCR2)**

[Table 18-15](#) describes the UMCR fields.

**Table 18-15. UMCR Field Descriptions**

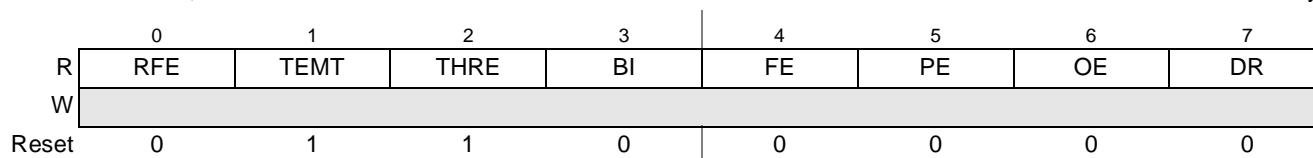
Bits	Name	Description
0–2	—	Reserved, should be cleared
3	LOOP	Local loopback mode 0 Normal operation. 1 Functionally, the data written to UTHR can be read from URBR of the same UART
4–7	—	Reserved

### 18.3.1.9 Line Status Registers (ULSR1 and ULSR2)

The ULSRs, shown in [Figure 18-11](#), 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.

Offset: 0x0\_4505, 0x0\_4605

Access: User read-only



**Figure 18-11. Line Status Register (ULSR1 and ULSR2)**

[Table 18-16](#) describes the ULSR fields.

**Table 18-16. ULSR Field Descriptions**

Bits	Name	Description
0	RFE	Receiver FIFO error. 0 Cleared when there are no errors in the receiver FIFO or on a read of the ULSR with no remaining receiver FIFO errors. 1 Set when one of the characters in the receiver FIFO encounters an error (framing, parity, or break interrupt).
1	TEMT	Transmitter empty 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. 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 0 UTHR is not empty. 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 0 Cleared when the ULSR is read or when a valid data transfer is detected (that is, STOP bit is received). 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 0 Cleared when ULSR is read or when a new character is loaded into the URBR from the receiver shift register. 1 Invalid STOP bit for receive data (only the first STOP bit is checked). In FIFO mode, FE 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 will receive the following new data.
5	PE	Parity error 0 Cleared when ULSR is read or when a new character is loaded into URBR. 1 Unexpected parity value encountered when receiving data. In FIFO mode, the character with the error is at the top of the FIFO.

Table 18-16. ULSR Field Descriptions (continued)

Bits	Name	Description
6	OE	Overrun error 0 Cleared when ULSR is read 1 Before URBR was read, it was overwritten with a new character. The old character is lost. 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 0 Cleared when URBR is read or when all of the data in the receiver FIFO is read. 1 A character was received in the URBR or the receiver FIFO.

### 18.3.1.10 Scratch Registers (USCR1 and USCR2)

USCR, shown in [Figure 18-12](#), are for debugging software or the DUART hardware. The USCRs do not affect the operation of the DUART.

Offset: 0x0\_4507, 0x0\_4607

Access: User read/write

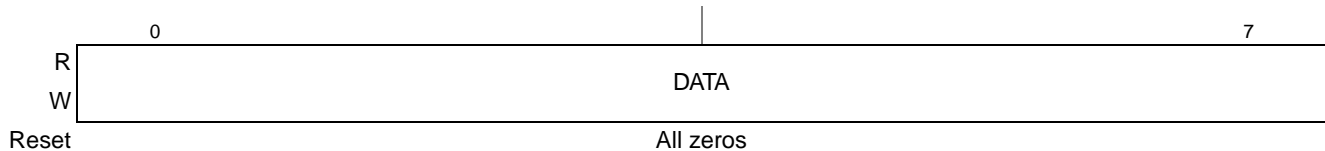


Figure 18-12. Scratch Register (USCR)

[Table 18-17](#) describes USCR fields.

Table 18-17. USCR Field Descriptions

Bits	Name	Description
0–7	DATA	Data

### 18.3.1.11 Alternate Function Registers (UAFR1 and UAFR2)

The UAFRs, shown in [Figure 18-13](#), allow software to write to both UART1 and UART2 registers simultaneously with the same write operation. The UAFRs also provide a means for the device's performance monitor to track the baud clock.

Offset: 0x0\_4502, 0x0\_4602

Access: User read/write



Figure 18-13. Alternate Function Register (UAFR)

Table 18-18 describes UAFR fields.

**Table 18-18. UAFR Field Descriptions**

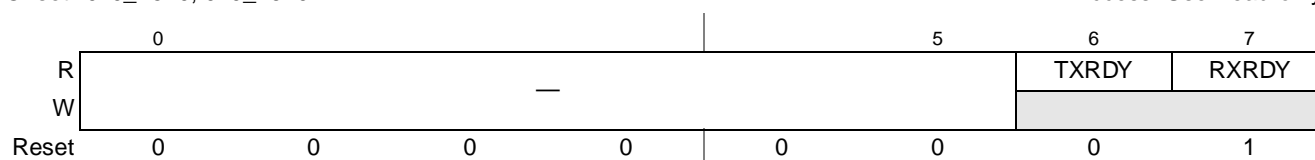
Bits	Name	Description
0–5	—	Reserved
6	BO	Baud clock select 0 The baud clock is not gated off. 1 The baud clock is gated off.
7	CW	Concurrent write enable 0 Disables writing to both UART1 and UART2. 1 Enables concurrent writes to corresponding UART registers. A write to a register in UART1 is also a write to the corresponding register in UART2 and vice versa.

### 18.3.1.12 DMA Status Registers (UDSR1 and UDSR2)

The DMA status registers (UDSRs), shown in Figure 18-14, return transmitter and receiver FIFO status and provide the ability to assist DMA data operations to and from the FIFOs.

Offset: 0x0\_4510, 0x0\_4610

Access: User read-only



**Figure 18-14. DMA Status Register (UDSR)**

Table 18-19 describes the fields of the UDSRs.

**Table 18-19. UDSR Field Descriptions**

Bits	Name	Description
0–5	—	Reserved
6	TXRDY	Transmitter ready. Reflects the status of the transmitter FIFO or the UTHR. The status depends on the DMA mode selected, which is determined by UFCR[DMS] and UFCR [FEN]. 0 The bit is cleared, as shown in Table 18-21. 1 This bit is set, as shown in Table 18-20.
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 UFCR[DMS] and UFCR [FEN]. 0 The bit is cleared, as shown in Table 18-23. 1 This bit is set, as shown in Table 18-22.

Table 18-20 and Table 18-21 show the set and cleared conditions for UDSR[TXRDY].

**Table 18-20. 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 18-21. 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 while the transmitter FIFO is not yet full.

Table 18-22 and Table 18-23 show the set and cleared conditions for UDSR[TRRDY].

**Table 18-22. 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 18-23. UDSR[RXRDY] Cleared**

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.

## 18.4 Functional Description

The communication channel provides a full-duplex asynchronous receiver and transmitter using an operating frequency derived from the system clock signal.

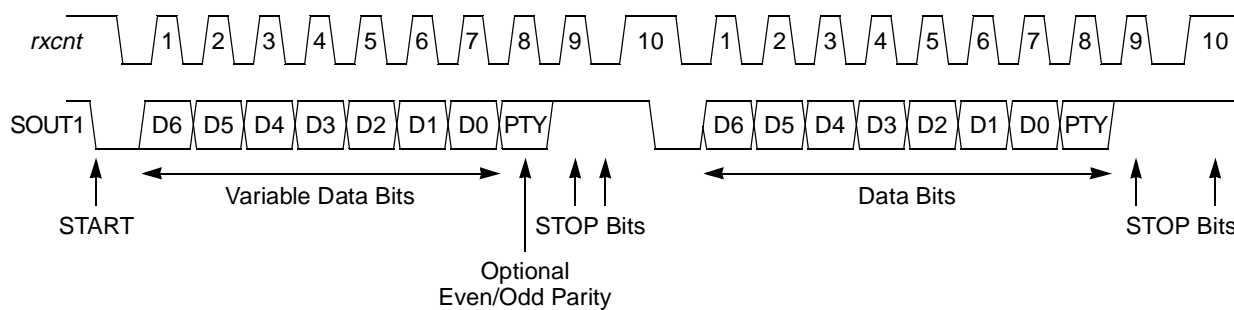


The transmitter accepts parallel data with a write access to UTHR. In FIFO mode, the data is placed directly into an internal transmitter shift register, or into the transmitter FIFO—see [Section 18.4.5, “FIFO Mode.”](#) The transmitting registers convert the data to a serial bit stream by inserting 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.

### 18.4.1 Serial Interface

The UART bus is a serial, full-duplex, point-to-point bus as shown in [Figure 18-15](#). 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 18-15. UART Bus Interface Transaction Protocol Example**

A standard UART bus transfer is composed of either three or four parts:

- START bit
- Data transfer (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.

#### 18.4.1.1 START Bit

A write to UTHR generates a START bit on the SOUT signal. [Figure 18-15](#) 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 ULCR. When the bus is idle, SOUT is high.

### 18.4.1.2 Data Transfer

Each data transfer contains 5, 6, 7, or 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 to 8 of the data bits previously written to the UTHR. The data bits are driven from the least- to the most-significant bits. After the parity and STOP bits, a new data transfer can begin if new data is written to UTHR.

### 18.4.1.3 Parity Bit

The user has the option of using even, odd, no parity, or stick parity (see [Section 18.3.1.7, “Line Control Registers \(ULCR1 and ULCR2\).”](#) Both the receiver and transmitter parity definitions must agree before transferring data. When receiving data, a parity error can occur if an unexpected parity value is detected (see [Section 18.3.1.9, “Line Status Registers \(ULSR1 and ULSR2\)”](#)).

### 18.4.1.4 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.

## 18.4.2 Baud-Rate Generator Logic

Each UART contains an independent programmable baud-rate generator, that is capable of taking the system 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:

$$\text{Baud rate} = (1/16) \times (\text{system clock frequency/divisor value}) \quad \text{Eqn. 18-1}$$

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 UAFR[BO]. This can be used to determine baud-rate errors.

### 18.4.3 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. 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. 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).

### 18.4.4 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.

#### 18.4.4.1 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.

#### 18.4.4.2 Parity Error

When unexpected parity values are encountered while receiving data, a parity error occurs and ULSR[PE] is set. 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.

#### 18.4.4.3 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.

### 18.4.5 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. 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.

### 18.4.5.1 FIFO Interrupts

In FIFO mode, the UIER[ERDAI] is set when a time-out interrupt occurs. A receive data time-out generates a maskable interrupt condition (through UIER[ERDAI]). See [Section 18.3.1.4, “Interrupt Enable Registers \(UIER1 and UIER2\).”](#)

UIIR indicates whether the FIFOs are enabled. UIIR[IID3] is set only 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 at least one character is in the receiver FIFO. The character time-out interrupt (controlled by UIIR[IID $n$ ]) is cleared when URBR is read. See [Section 18.3.1.5, “Interrupt ID Registers \(UIIR1 and UIIR2\).”](#)

UIIR[FE] indicates whether FIFO mode is enabled.

### 18.4.5.2 DMA Mode Select

UDSR[RXRDY] reflects the status of the receiver FIFO or URBR. In mode 0 (UFCR[DMS] is cleared), UDSR[RXRDY] is cleared when at least one character is in the receiver FIFO or URBR; it is set when there are no more characters in the receiver FIFO or URBR. This occurs regardless of the UFCR[FEN] setting. 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; it is set when there are no more characters in the receiver FIFO.

UDSR[TXRDY] 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; it is set after the first character is loaded into the transmitter FIFO or UTHR. This occurs regardless of the UFCR[FEN] setting. 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; it is set when the transmitter FIFO is full.

See [Section 18.3.1.12, “DMA Status Registers \(UDSR1 and UDSR2\).”](#) for a complete description of the UDSR[RXRDY] and UDSR[TXRDY] bits.

### 18.4.5.3 Interrupt Control Logic

An interrupt is active when DUART interrupt ID register bit 7 (UIIR[IID0]), is cleared. UIER is used to mask specific interrupt types. See [Section 18.3.1.4, “Interrupt Enable Registers \(UIER1 and UIER2\).”](#)

When the interrupts are disabled in UIER, polling software can not use UIIR[IID0] to determine whether the UART is ready for service. Software must monitor the appropriate ULSR bit. UIIR[IID0] can be used for polling if the interrupts are enabled in UIER.

## 18.5 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 must be 1 byte wide. Reads and writes to these registers must be byte-length 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 19

## Serial Peripheral Interface

### 19.1 Overview

The serial peripheral interface (SPI) allows the device to exchange data between other PowerQUICC family chips, the MC68360, MC68302, M68HC11, and M68HC05 microcontroller families, and other family devices. The SPI can be used to communicate with peripheral devices such as EEPROMs, real-time clocks, A/D converters, and ISDN devices.

The SPI is a full-duplex, synchronous, character-oriented channel that supports a four-wire interface (receive, transmit, clock, and slave select). The SPI block 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 SPI baud rate generator in master mode or externally in slave mode. During an SPI transfer, data is sent and received simultaneously.

The SPI receiver and transmitter are double-buffered, as shown in the block diagram in Figure 19-1. This gives an effective FIFO size (latency) of 2 characters. The SPI's MSB/LSB is shifted out first. When the SPI is disabled in the SPI mode register (SPMODE[EN] = 0), it consumes little power.

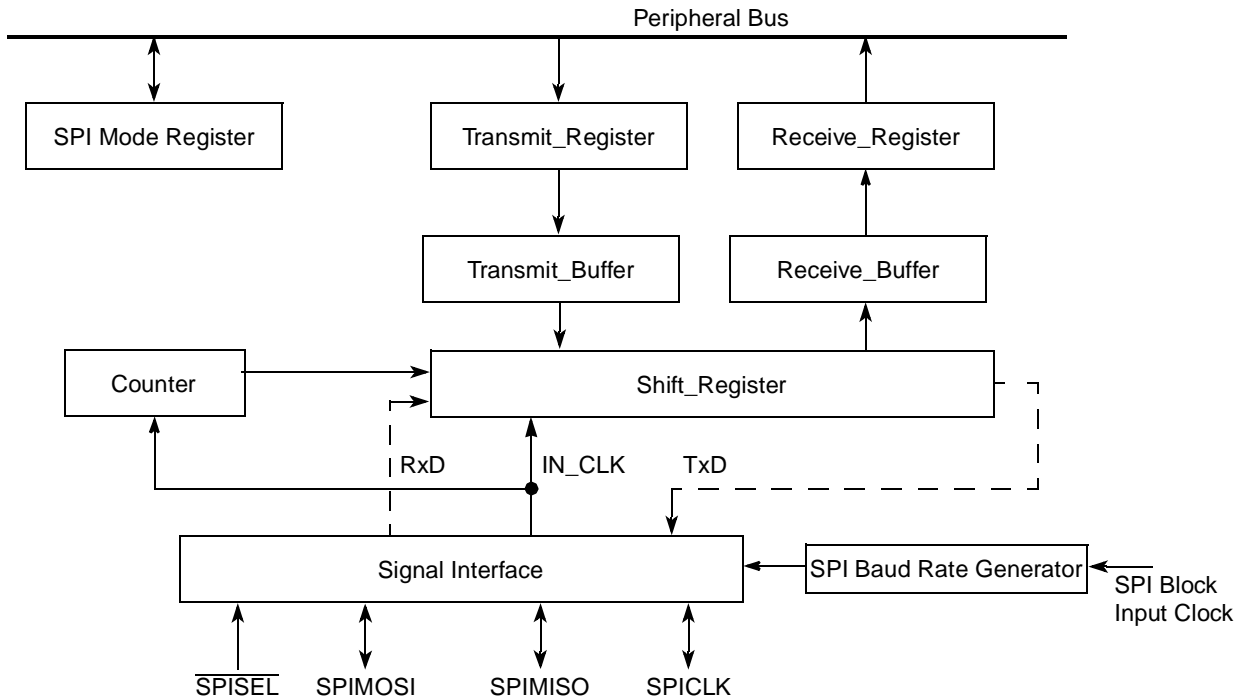


Figure 19-1. SPI Block Diagram

## 19.1.1 Features

The major features of the SPI are listed as follows:

- Four-signal interface (SPIMOSI, SPIMISO, SPICLK, and  $\overline{\text{SPISEL}}$ )
- Full-duplex operation
- Works with 32-bit data characters or with a range from 4-bit to 16-bit data characters
- Supports back-to-back character transmission and reception
- Supports reverse data mode for 8/16/32 character length
- Supports master SPI mode
- Supports multiple-master environment
- Maximum clock rate is (input clock rate/4) in master mode; (input clock rate/2) in slave mode
- Independent programmable baud rate generator
- Programmable clock phase and polarity
- Local loopback capability for testing
- Open-drain outputs support multiple-master configuration

## 19.1.2 SPI Transmission and Reception Process

Because the SPI 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 SPI transmission session, from the first character written to the SPITD register to the last character transmitted following the setting of SPCOM[LST]. See [Section 19.3.1.4, “SPI Command Register \(SPCOM\),”](#) for more information.

The core receives data by reading the SPI receive data hold register (SPIRD). The SPI then clears the not empty SPIE[NE] to free up the SPIRD register for the next receive operation. The core transmits data by writing it into the SPI transmit data hold register (SPITD). The SPI then clears the not full (NF) bit in the SPI event register (SPIE) to indicate that the SPITD register contains a character for transmission. When the next character to be transmitted is going to be the final one in the current frame, the core sets SPCOM[LST], and then writes the final character to SPITD.

The SPI core handshake protocol can be implemented by either using polling or interrupts. When using a polling, 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 SPI serial channel frequency. When using the interrupt mechanism, setting either the not full (NF) or not empty (NE) bits of SPIE causes an interrupt to the processor core. The core then reads SPIE and acts accordingly. The three basic modes of operation for transmitting and receiving are master, slave and multiple-master.

### NOTE

When both NE and NF bits are set, the processor core should read the received data before transmitting new data.

The SPMODE[LEN] determines the character length sent by the hardware. The core is responsible for any bit manipulation to pack/unpack data into the appropriate character length. See the SPMODE[LEN] description in [Table 19-4](#) for more information.





The SPI sets SPIE[NF] to issue a maskable interrupt to the interrupt controller whenever its transmit buffer is not full. It also sets the NF bit after sending the last word. In response, the core should read the exception flags that relate to the last word. The SPI sets SPIE[NE] to issue a maskable interrupt to the interrupt controller whenever the receiver buffer has been filled with data.

### 19.1.3.2 SPI as a Slave Device

In slave mode, the SPI receives messages from an SPI master and sends a simultaneous reply. The slave's  $\overline{\text{SPISEL}}$  must be asserted before Rx clocks are recognized. Once  $\overline{\text{SPISEL}}$  is asserted, SPICLK becomes an input from the master to the slave. SPICLK can be any frequency from DC to input clock/2.

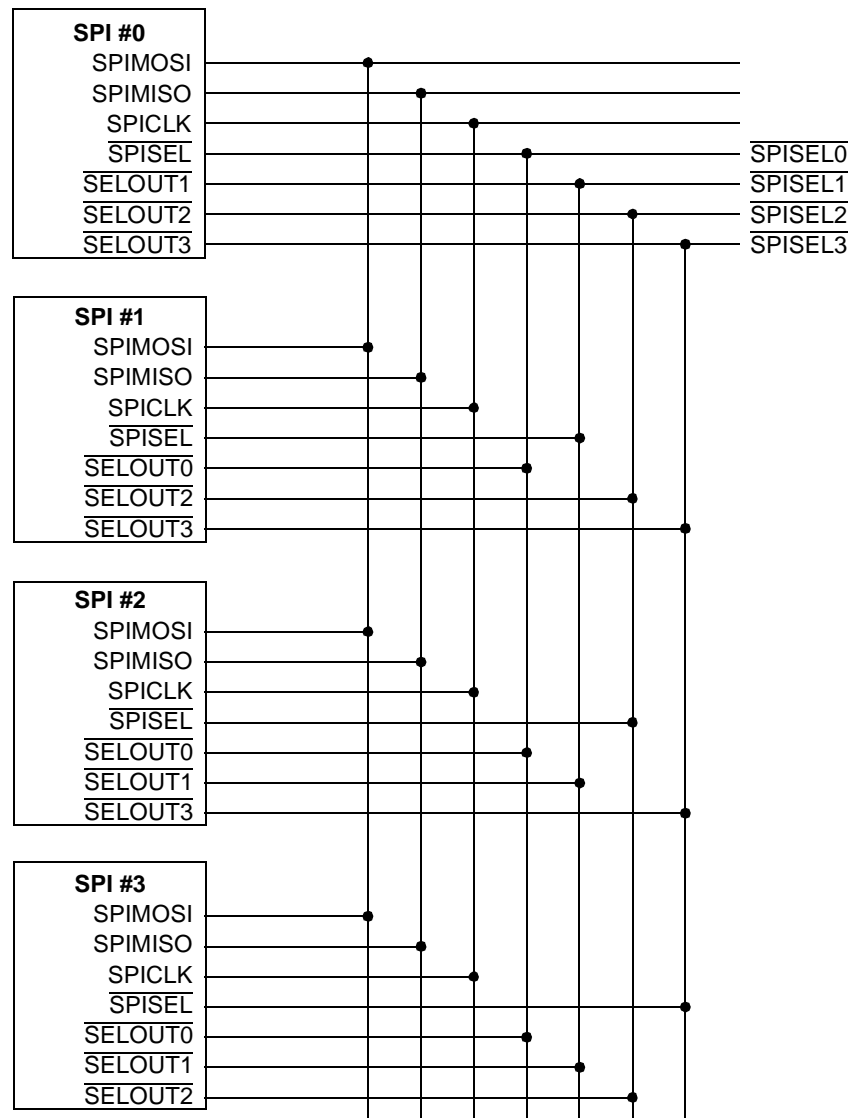
To prepare for data transfers, the core writes data to be sent into the SPITD register. Once  $\overline{\text{SPISEL}}$  is asserted, the slave shifts data out from SPIMISO and in through SPIMOSI. The SPI sets the NF bit of the SPIE register and a maskable interrupt is issued when a full buffer finishes receiving and sending or after an error. The SPI continues reception until  $\overline{\text{SPISEL}}$  is negated.

Transmission continues until no more data is available or  $\overline{\text{SPISEL}}$  is negated. Transmission continues once  $\overline{\text{SPISEL}}$  is reasserted and SPICLK begins toggling. After the characters in the buffer are sent, the SPI sends one as long as  $\overline{\text{SPISEL}}$  remains asserted.

### 19.1.3.3 SPI in Multiple-Master Operation

The SPI can operate in a multiple-master environment in which all SPI devices are connected to the same bus. In this configuration, the SPIMOSI, SPIMISO, and SPICLK signals of all SPIs are shared; but the  $\overline{\text{SPISEL}}$  inputs are connected separately, as shown in [Figure 19-3](#). Only one SPI device can act as master at a time—all others must be slaves. When a SPI is configured as a master, if its  $\overline{\text{SPISEL}}$  input is asserted, a multiple-master error occurs because more than one SPI device is a bus master. The SPI sets SPIE[MME] in the SPI event register and a maskable interrupt is issued to the core. It also disables SPI operation and

the output drivers of the SPI signals. The core must clear SPMODE[EN], correct the problems, and clear SPIE[MME] before the SPI can be used again.



**Notes:**

1. All signals are open-drain.
2. For a multiple-master configuration with more than two masters,  $\overline{\text{SPISEL}}$  and SPIE[MME] do not detect all possible conflicts.
3. It is the responsibility of software to arbitrate for the SPI bus (with token passing, for example).
4.  $\overline{\text{SELOUT}}_x$  signals are implemented in software with general-purpose I/O signals.

**Figure 19-3. Multiple-Master Configuration**

The maximum sustained data rate that the SPI supports is input clock/50. However, the SPI can transfer a single character at much higher rates—input clock/4 in master mode and input clock/2 in slave mode. Gaps should be inserted between multiple characters to keep from exceeding the maximum sustained data rate.

## 19.2 External Signal Descriptions

The SPI's four wire interface consists of transmit, receive, clock, and slave select.

### 19.2.1 Overview

Table 19-1 lists signal properties.

**Table 19-1. Signal Properties**

Name	Function	Reset	Pull Up
SPIMISO	Master input slave output	—	Required in open drain mode
SPIMOSI	Master output slave input	—	Required in open drain mode
SPICLK	Input/output serial clock connected to the other SPICLK	—	Required in open drain mode
$\overline{\text{SPISSEL}}$	SPI slave select	—	Required in open drain mode

### 19.2.2 Detailed Signal Descriptions

Table 19-2 describes the signals in detail.

**Table 19-2. Detailed Signal Descriptions**

Signal	I/O	Description
SPIMISO	I/O	Master input slave output
		<b>State Meaning</b> Asserted—The data that has been transmitted/received from/to the SPI (depends if master or slave mode) is high Negated—The data that has ben transmitted/received from/to the SPI (depends if master or slave mode) is low
		<b>Timing</b> Assertion—According to the SPICLK assertion/negation/in the middle of phase (depends on SPMODE) Negation—According to the SPICLK assertion/negation/in the middle of phase (depends on SPMODE)
SPIMOSI	I/O	Master output slave input
		<b>State Meaning</b> Asserted—The data that has been transmitted/received from/to the SPI (depends if master or slave mode) is high Negated—The data that has ben transmitted/received from/to the SPI (depends if master or slave mode) is low
		<b>Timing</b> Assertion—According to the SPICLK assertion/negation/in the middle of phase (depends on SPMODE) Negation—According to the SPICLK assertion/negation/in the middle of phase (depends on SPMODE)

Table 19-2. Detailed Signal Descriptions (continued)

Signal	I/O	Description	
SPICLK	I/O	Serial clock in or serial clock out for slave or master mode respectively	
		<b>State Meaning</b>	Assertion/Negation according to SPMODE[PM, DIV16] register rate configuration
		<b>Timing</b>	Assertion/Negation—during frame reception/transmission
$\overline{\text{SPISEL}}$	I	SPI slave select	
		<b>State Meaning</b>	Asserted—In slave mode declares the slave has been selected for the coming frame. In master mode assertion causes MME multiple-master error. Negated—In slave mode means the specific SPI has not been selected. In master mode needs to be negated for regular operation.
		<b>Timing</b>	Assertion—In slave mode along with the data from the slave Negation—In slave mode with the end of the frame (according to SPMODE[LEN]). In master mode before data is first written to SPITD and remains constant.

The SPI can be configured as a slave or a master in single- or multiple-master environments mode. The master SPI generates the transfer clock SPICLK using the SPI baud rate generator (BRG). The SPI BRG takes its input from input clock, which is generated in the device clock synthesizer.

SPICLK is a gated clock, active only during data transfers. Four combinations of SPICLK phase and polarity can be configured with the clock invert (SPMODE[CI]) and clock phase (SPMODE[CP]) register bits. SPI signals can also be configured as open-drain to support a multiple-master configuration in which a shared SPI signal is driven by the device or an external SPI device.

The SPI master-in slave-out SPIMISO signal acts as an input for master devices and as an output for slave devices. Conversely, the master-out slave-in SPIMOSI signal is an output for master devices and an input for slave devices. The dual functionality of these signals allows the SPIs in a multiple-master environment to communicate with one another using a common hardware configuration.

- When the SPI is a master, SPICLK is the clock output signal that shifts received data in from SPIMISO and transmitted data out to SPIMOSI. SPI masters must output a slave select signal to enable SPI slave devices using a separate general-purpose I/O signal. Assertion of the  $\overline{\text{SPISEL}}$  while the SPI is configured as a master causes an error.
- When the SPI is a slave, SPICLK is the clock input that shifts received data in from SPIMOSI and transmitted data out through SPIMISO.  $\overline{\text{SPISEL}}$  is the enable input to the SPI slave. In a multiple-master environment,  $\overline{\text{SPISEL}}$  (always an input) is also used to detect an error when more than one master is operating.

### 19.3 Memory Map/Register Definition

Table 19-3 shows the memory mapped registers of the SPI 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



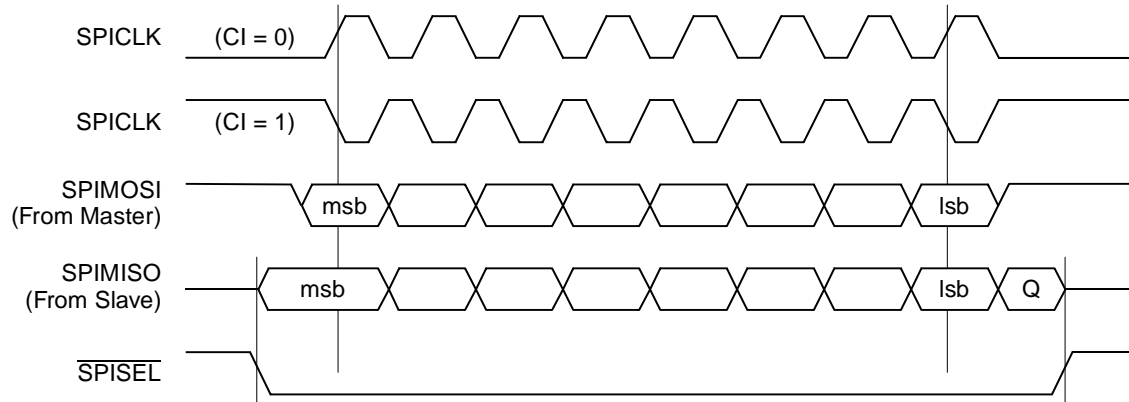
Table 19-4. SPMODE Field Descriptions (continued)

Bits	Name	Description
2	CI	Clock invert. Inverts SPI clock polarity. See <a href="#">Figure 19-5</a> and <a href="#">Figure 19-6</a> for more information 0 The inactive state of SPICLK is low. 1 The inactive state of SPICLK is high.
3	CP	Clock phase. Selects the transfer format. See <a href="#">Figure 19-5</a> and <a href="#">Figure 19-6</a> for more information. 0 SPICLK starts toggling at the middle of the data transfer. 1 SPICLK starts toggling at the beginning of the data transfer.
4	DIV16	Divide by 16. Selects the clock source for the SPI baud rate generator (SPI BRG) when configured as an SPI master. In slave mode, SPICLK is the clock source. 0 The SPI block input clock is the input to the SPI BRG. 1 The SPI block input clock/16 is the input to the SPI BRG. In slave mode, this bit must be cleared.
5	REV	Reverse data mode for 8-/16-/32-bit character length only (see <a href="#">Section 19.3.1.6.1</a> , “Reverse Mode SPMODE[REV] Examples.”) 0 LSB sent/received first (for data LEN < 32 the data is located at the lower half-word LSB) 1 MSB sent/received first
6	M/S	Master/slave. Selects master or slave mode. 0 The SPI is a slave. 1 The SPI is a master.
7	EN	Enable SPI. Any other bits in SPMODE must not change when EN is set. 0 The SPI is disabled. The SPI is in a idle state and consumes minimal power. The SPI BRG is not functioning and the input clock is disabled. 1 The SPI is enabled. <b>Note:</b> The SPI controller requires a minimal gap of at least 10 input clocks between disabling the SPI and re-enabling. This minimal gap is sufficient provided that SPMODE[PM] and SPMODE[DIV16] are cleared during the time in which SPMODE[EN] is cleared.
8–11	LEN	Character length in bits per character. LEN can be either 32-bits, or 4- to 16-bits that are shown as follows: 0000 32-bit characters 0001–0010 Reserved, causes erratic behavior. 0011 4-bit characters ... 1111 16-bit characters The TX and RX registers (SPITD, SPIRD) hold 32 bits at a time. A character length of 32 bits fills the TX and RX registers; therefore, all of the bits in these registers are valid. However, if the character length selected by LEN is equal or less than 16 bits, then the valid bits will reside in the lower half-word of the transmit and receive registers. For example, if the character length is set to 16 bits than the valid bits will be 16–31, if the character length is set to 5 bits, the valid bits are 16–20. Note that the transmit and receive registers each can hold only one character regardless of the character length.
12–15	PM	Prescale modulus select. Specifies the divide ratio of the prescale divider in the SPI clock generator. The SPI baud rate generator clock source (either input clock or input clock divided by 16, depending on DIV16 bit) is divided by $4 \times ([PM] + 1)$ , a range from 4 to 64. The clock has a 50% duty cycle. For example, if the prescale modulus is set to PM = 0011 and DIV16 is set, the system/SPICLK clock ratio will be $16 \times (4 \times (0011 + 1)) = 256$ . In slave mode, this field must be cleared.
16–18	—	Reserved. Should be cleared.

Table 19-4. SPMODE Field Descriptions (continued)

Bits	Name	Description
19	OD	Open drain mode. 0 All output pins are configured to normal mode. 1 All output pins are configured to open drain mode.
20–31	—	Reserved. Should be cleared.

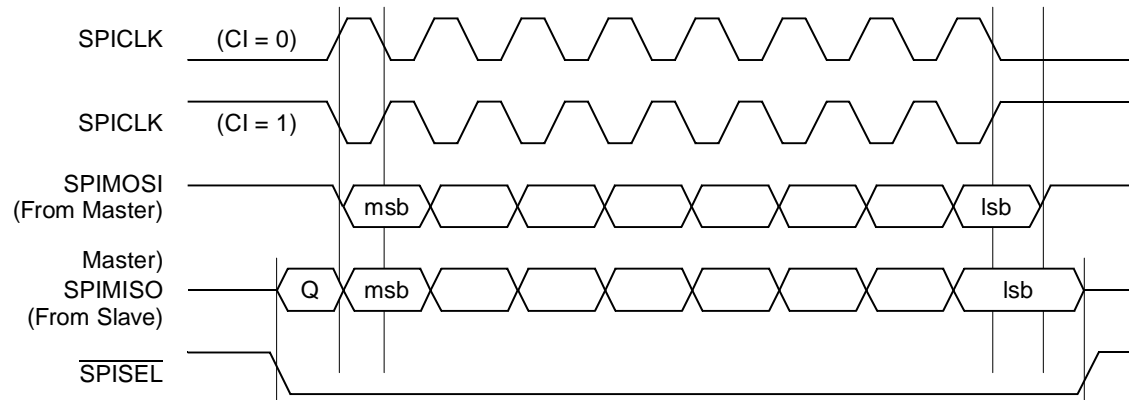
Figure 19-5 shows the SPI transfer format in which SPICLK starts toggling in the middle of the transfer (SPMODE[CP] = 0).



NOTE: Q = Undefined signal.

Figure 19-5. SPI Transfer Format with SPMODE[CP] = 0

Figure 19-6 shows the SPI transfer format in which SPICLK starts toggling at the beginning of the transfer (SPMODE[CP] = 1).



NOTE: Q = Undefined signal.

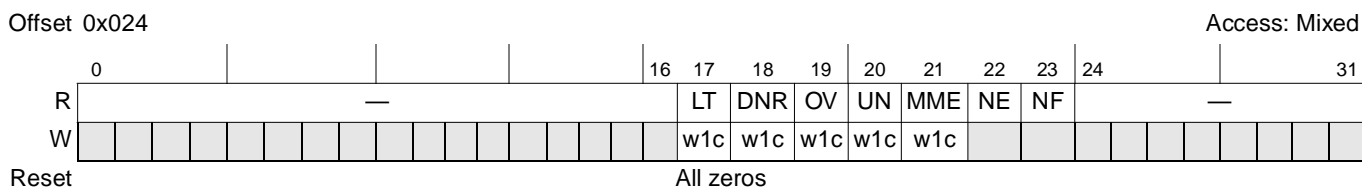
Figure 19-6. SPI Transfer Format with SPMODE[CP] = 1

### 19.3.1.2 SPI Event Register (SPIE)

The SPI event register (SPIE) generates interrupts and reports events recognized by the SPI. When an event is recognized, the SPI sets the corresponding SPIE bit. Most SPIE bits can be cleared by writing a



'1'. Writing '0' has no effect. Setting a bit in the SPI 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. Figure 19-7 shows SPI event register.



**Figure 19-7. SPIE—SPI Event Register Definition**

Table 19-5 describes the SPIE fields.

**Table 19-5. SPIE Field Descriptions**

Bits	Name	Description
0–16	—	Reserved, should be cleared.
17	LT	Last character was transmitted. The last character is transmitted and new data can be written to SPID for further transmission.
18	DNR	<b>Note:</b> Data not ready. In slave mode only when $\overline{\text{SPISEL}}$ is asserted before data is ready in the SPI, IDLE is sent on the line and UN bit is also asserted the SPI should be disable to restart its operation.
19	OV	Slave/master overrun. Indicates whether an overrun has occurred during reception. In case of overrun the SPI continues transmission/reception process while reporting overrun for the missing characters.
20	UN	Slave underrun. Indicates whether the SPI transmitter did not have data to transmit on time, and therefore, whether IDLE was sent on the line. Valid only in slave mode (SPMODE[M/S]) = 0. In master mode (SPMODE[M/S]) = 1) if the SPI's transmitter has no valid data to transmit the SPICLK stop toggling and transmission/reception is frozen (no underrun is reported), when data is written to the SPITD the transmission resumes.
21	MME	Multiple-master error. Set when $\overline{\text{SPISEL}}$ is asserted externally while the SPI is in master mode. Note that the MME error can occur in loopback mode.
22	NE	Not empty. When set Indicates that SPIRD contains a received character. 0 The receiver is empty 1 The receiver has valid received data and indications about LST (command register) and OV (SPIE).The core is free to read the content of the receiver. Reading the receiver SPIRD clears NE if no more data is available.
23	NF	Not full. Indicates whether SPITD is not in use and a new character can be written to it by the core. 0 The transmitter is full. 1 The transmitter is not full. The core is free to write to the transmitter. NF must be clear to enable the transmission of another character (writing to the transmitter clears NF)
24–31	—	Reserved. Should be cleared.

### 19.3.1.3 SPI Mask Register (SPIM)

The SPI mask register (SPIM), shown in Figure 19-8, enables/masks interrupts for events that are recognized by the SPI. When an event is recognized, the SPI sets the corresponding SPIE bit. Setting a

SPIM bit enables and clearing a SPIM bit masks the corresponding interrupt. Unmasked SPIE bits must be cleared before the core clears its internal interrupt requests.

Offset 0x028

Access: Read/write

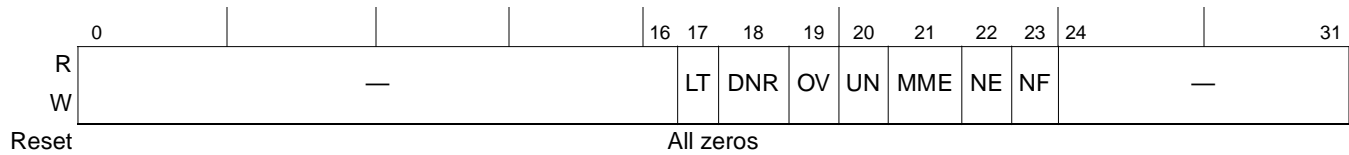


Figure 19-8. SPIM—SPI Mask Register Definition

Table 19-6 describes the SPIM fields.

Table 19-6. SPIM Field Descriptions

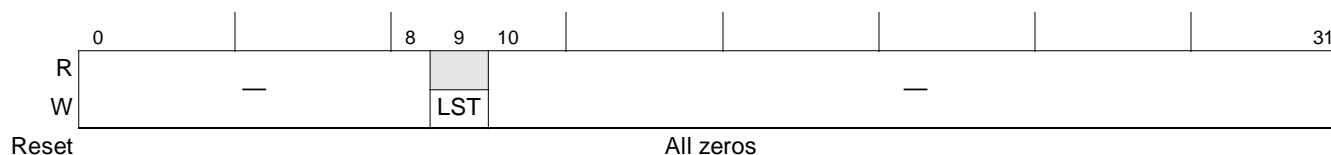
Bits	Name	Description
0–16	—	Reserved, should be cleared.
17	LT	Last character transmitted 0 LT event will not cause an SPI interrupt 1 LT event causes an SPI interrupt
18	DNR	In slave mode data not ready 0 Slave DNR event will not cause an SPI interrupt <b>Note:</b> 1 Slave DNR event causes an SPI interrupt
19	OV	Slave/Master Overrun interrupt mask 0 Slave/Master Overrun event will not cause an SPI interrupt 1 Slave/Master Overrun event causes an SPI interrupt
20	UN	Slave Underrun interrupt mask 0 Slave Underrun event will not cause an SPI interrupt 1 Slave Underrun event causes an SPI interrupt
21	MME	Multimaster error interrupt mask 0 Multimaster error event will not cause an SPI interrupt 1 Multimaster error event causes an SPI interrupt
22	NE	Not Empty interrupt mask 0 Not Empty event will not cause an SPI interrupt 1 Not Empty event causes an SPI interrupt
23	NF	Not Full interrupt mask 0 Not Full event will not cause an SPI interrupt 1 Not Full event causes an SPI interrupt
24–31	—	Reserved, should be cleared.

### 19.3.1.4 SPI Command Register (SPCOM)

The SPI command register (SPCOM), shown in [Figure 19-9](#), is used to end SPI operation.

Offset 0x02C

Access: Write only



**Figure 19-9. SPI Command Register Definition**

[Table 19-7](#) describes the SPCOM fields.

**Table 19-7. SPCOM Field Descriptions**

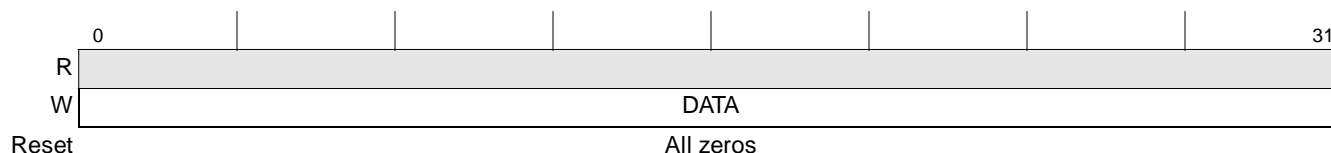
Bits	Name	Description
0–8	—	Reserved, should be cleared.
9	LST	This bit represents the last character. Should be set before the last character is written to the SPITD. This results in SPIE[LT] being set when the character is fully transmitted and by that gives indication about the frame being fully transmitted. 0 This character is not the last character of the frame 1 This character is the last character of the frame
10–31	—	Reserved, should be cleared.

### 19.3.1.5 SPI Transmit Data Hold Register (SPITD)

SPITD holds the character to be transmitted. The number of bits in each character is specified by SPMODE[LEN]. Each time SPIE[NF] is set, the core can write another character of data to SPITD, if there is no error indication in the SPIE. At the end of the frame the core should set SPCOM[LST] and prepare the last character of data. [Figure 19-10](#) shows the SPI transmit data hold register.

Offset 0x030

Access: Write only



**Figure 19-10. SPI Transmit Data Hold Register Definition**

[Table 19-8](#) shows the field descriptions of the SPI transmit data hold register.

**Table 19-8. SPI Transmit Data Hold Field Descriptions**

Bits	Name	Description
0–31	DATA	These bits are the data to be sent.

### 19.3.1.6 SPI Receive Data Hold Register (SPIRD)

SPIRD, shown in Figure 19-11, is used to receive a character of data from the SPI channel. Each time SPIE[NE] is set, the core can read SPIRD.

Offset 0x034

Access: Read-only

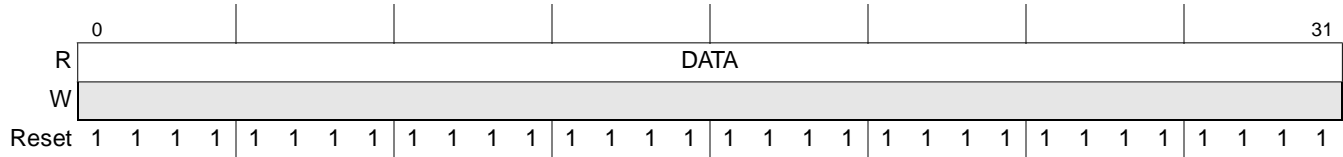


Figure 19-11. SPI Receive Data Hold Register Definition

Table 19-9 shows the field descriptions of the SPI receive data hold register.

Table 19-9. SPI Receive Data Hold Field Descriptions

Bits	Name	Description
0–31	DATA	Received data. These bits are the received data from the SPI bus.

#### 19.3.1.6.1 Reverse Mode SPMODE[REV] Examples

In reverse data mode (SPMODE[REV] = 1) and regular data mode (SPMODE[REV] = 0), the data is placed in the SPIRD after reception is completed as described below for character length of 8 bits (SPMODE[LEN] = 7).

Offset 0x036

Access: Read-only

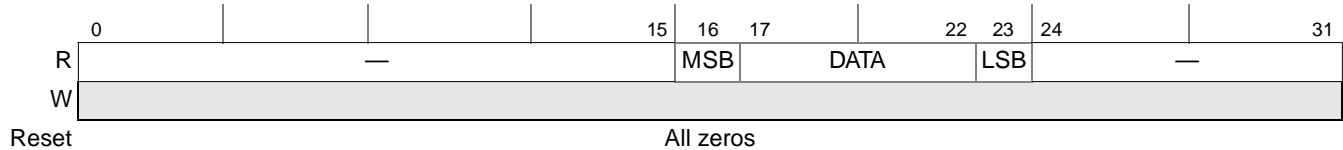


Figure 19-12. Example SPMODE[REV] = 0 SPMODE[LEN] = 7 LSB Sent First

Offset 0x036

Access: Read-only

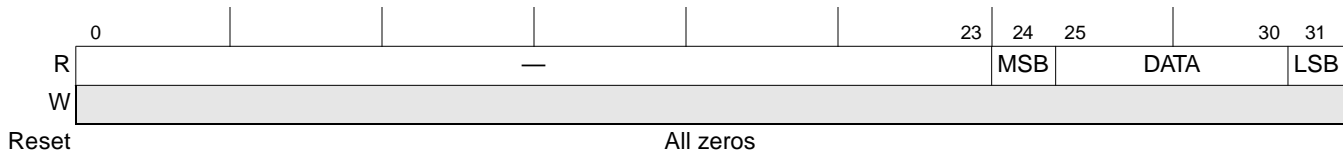


Figure 19-13. Example SPMODE[REV] = 1 SPMODE[LEN] = 7 MSB Sent First

Offset 0x036

Access: Read-only

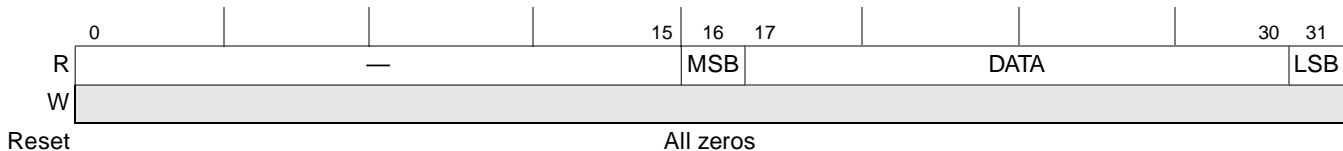


Figure 19-14. Example SPMODE[REV] = 1 SPMODE[LEN] = 15 MSB Sent First

Downloaded from [Elcodis.com](http://Elcodis.com) electronic components distributor

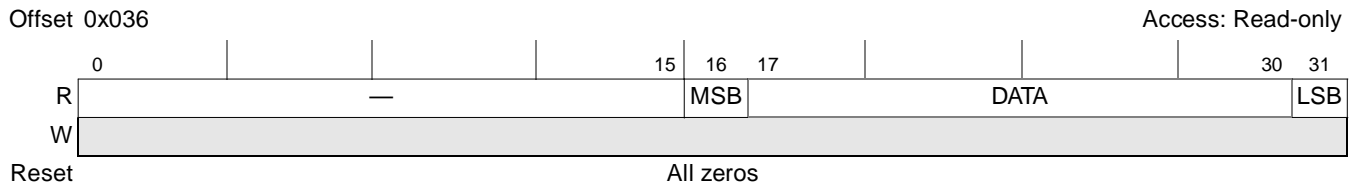


Figure 19-15. Example SPMODE[REV] = 0 SPMODE[LEN] = 15 LSB Sent First

## 19.4 Initialization/Application Information

The following sections describe programming examples of the SPI master and slave.

### 19.4.1 SPI Master Programming Example

The following sequence initializes the SPI to run at a high speed in master mode:

1. Configure a parallel I/O signal to operate as the SPI select output signal if needed.
2. Write 0xFFFF\_FFFF to SPIE to clear any previous events. Configure SPIM to enable all desired SPI interrupts.
3. Configure SPMODE to enable normal operation (not loopback), master mode, SPI enabled, character length, and the fastest speed possible.
4. Write the first character to be sent to SPITD.

### 19.4.2 SPI Slave Programming Example

The following is an example initialization sequence to follow when the SPI is in slave mode. It is very similar to the SPI master example, except that  $\overline{\text{SPISEL}}$  is used instead of a general-purpose I/O signal.

1. Write 0xFFFF\_FFFF to SPIE to clear any previous events.
2. Configure SPIM to enable all desired SPI interrupts.
3. Configure SPMODE to enable normal operation (not loopback), slave mode, SPI enabled, and characters length.
4. Write the first data to be sent to SPITD, to enable the SPI to be ready once the master begins to transfer.



# Chapter 20

## JTAG/Testing Support

### 20.1 Overview

The device provides a JTAG (Joint Test Action Group) interface to facilitate boundary-scan testing. The JTAG interface complies to the IEEE 1149.1 boundary-scan specification. For additional information about JTAG operations, refer to the IEEE 1149.1 specification.

The JTAG interface consists of a set of five signals, three JTAG registers (see Section 20.3, “JTAG Registers and Scan Chains,”) and a test access port (TAP) controller, described in the following sections. A block diagram of the JTAG interface is shown in Figure 20-1.

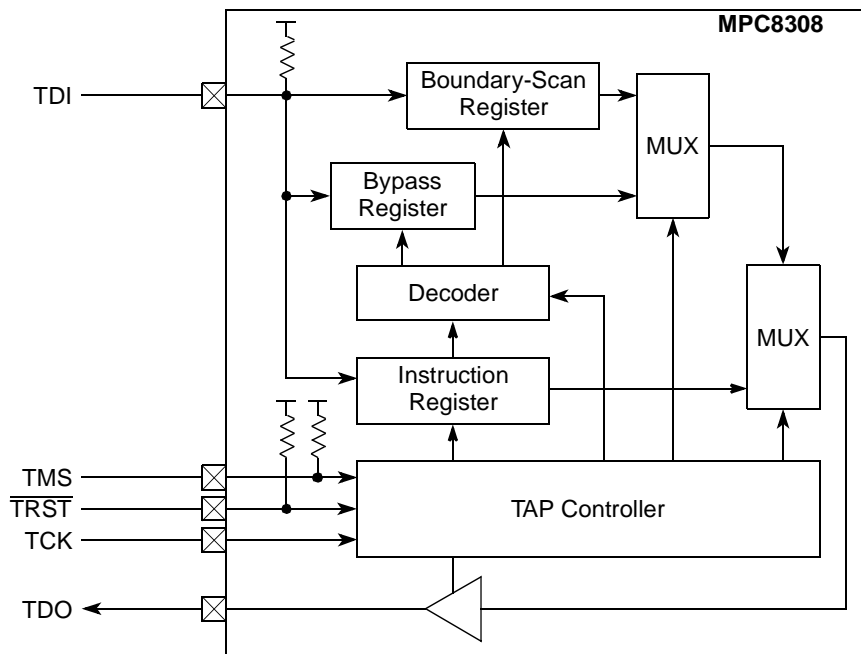


Figure 20-1. JTAG Interface Block Diagram

### 20.2 JTAG Signals

The device provides the following five dedicated JTAG signals:

- Test data input (TDI)
- Test data output (TDO)
- Test mode select (TMS)
- Test reset ( $\overline{\text{TRST}}$ )

- Test clock (TCK)

The TDI and TDO signals input and output all instructions and data to the JTAG scan registers. JTAG operations are controlled by the TAP controller through the TMS and TCK signals. Boundary-scan data is latched by the TAP controller on the rising edge of the TCK signal. The  $\overline{\text{TRST}}$  signal is specified as optional by the IEEE 1149.1 specification, and is used to reset the TAP controller asynchronously. The assertion of the  $\overline{\text{TRST}}$  signal at power-on reset ensures that the JTAG logic does not interfere with the normal operation of the device.

## 20.2.1 External Signal Descriptions

The JTAG signals are summarized in [Table 20-1](#).

**Table 20-1. JTAG Test Signals Summary**

Name	Description	Functional Block	Function	Reset Value	I/O
TCK	Test clock	Debug	Clock for JTAG testing.	—	I
TDI	Test data input		Serial input for instructions and data to the JTAG test subsystem. Internally pulled up.	—	I
TDO	Test data output		Serial data output for the JTAG test subsystem. High impedance except when scanning out data.	High impedance	O
TMS	Test mode select		Carries commands to the TAP controller for boundary scan operations. Internally pulled up.	—	I
$\overline{\text{TRST}}$	Test reset		Resets the TAP controller asynchronously. Internally pulled up.	—	I

[Table 20-2](#) shows detailed descriptions of the JTAG test signals.

**Table 20-2. JTAG Test—Detailed Signal Descriptions**

Signal	I/O	Description	
TCK	I	JTAG test clock.	
		<b>State Meaning</b>	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.
		<b>Timing</b>	See IEEE 1149.1 specification for more details.
TDI	I	JTAG test data input.	
		<b>State Meaning</b>	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.
		<b>Timing</b>	See IEEE 1149.1 specification for more details.



Table 20-2. JTAG Test—Detailed Signal Descriptions (continued)

Signal	I/O	Description	
TDO	O	JTAG test data output.	
		<b>State Meaning</b>	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.
		<b>Timing</b>	See IEEE 1149.1 specification for more details.
TMS	I	JTAG test mode select.	
		<b>State Meaning</b>	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.
		<b>Timing</b>	See IEEE 1149.1 specification for more details.
TRST	I	JTAG test reset.	
		<b>State Meaning</b>	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 device. An unterminated input appears as a high signal level to the test logic due to an internal pull-up resistor. Negated— Normal operation.
		<b>Timing</b>	See IEEE 1149.1 specification for more details.

## 20.3 JTAG Registers and Scan Chains

The bypass, boundary-scan, and instruction JTAG registers and their associated scan chains are mandatory for conformity to the IEEE 1149.1 specification, as follows.

- Bypass register

The bypass register is a single-stage register used to bypass the boundary-scan latches of the device during board-level boundary-scan operations involving components other than the device. The use of the bypass register reduces the total scan string size of the boundary-scan test.

- Boundary-scan registers

The JTAG interface provides a chain of registers dedicated to boundary-scan operations. To be JTAG-compliant, these registers cannot be shared with any functional registers of the device. The boundary-scan register chain includes registers controlling the direction of the input/output drivers, in addition to the registers reflecting the signal value received or driven.

The boundary-scan registers capture the input or output state of the device's signals during a Capture\_DR TAP controller state. When a data scan is initiated following the Capture\_DR state, the sampled values are shifted out through the TDO output while new boundary-scan register values are shifted in through the TDI input. At the end of the data scan operation, the boundary-scan registers are updated with the new values during an update\_DR TAP controller state.

- Instruction register

The 8-bit JTAG instruction register serves as an instruction and status register. As TAP controller instructions are scanned in through the TDI input, the TAP controller status bits are scanned out through the TDO output.

- TAP controller

The device provides a standard JTAG TAP controller that controls instruction and data scan operations. The TMS signal controls the state transitions of the TAP controller.

# Chapter 21

## General Purpose I/O (GPIO)

### 21.1 Introduction

This chapter describes the general-purpose I/O module, including pin descriptions, register settings, and interrupt capabilities. [Figure 21-1](#) shows the block diagram of the GPIO module.

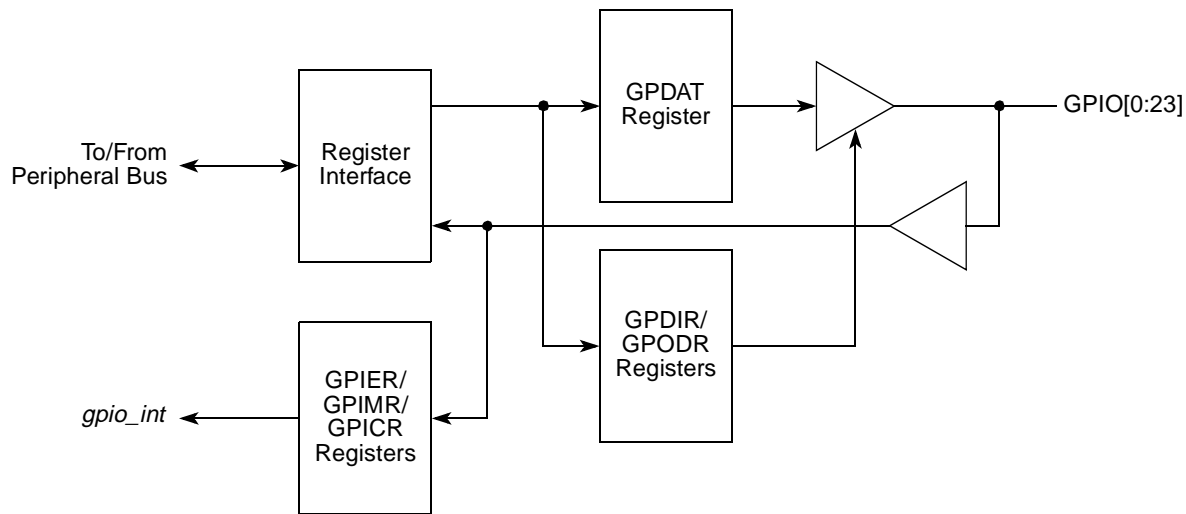


Figure 21-1. GPIO Module Block Diagram

#### 21.1.1 Overview

The GPIO module supports 16 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.

#### 21.1.2 Features

The GPIO unit implements the following features:

- 16 input/output ports
- Some ports have dedicated processor signals. Others are multiplexed together with other functional signals. See [Chapter 2, “Signal Descriptions.”](#)
- 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.

## 21.2 External Signal Description

The following section provides information about GPIO signals.

### 21.2.1 Signals Overview

Table 21-1 provides detailed descriptions of the external GPIO signals.

**Table 21-1. GPIO—Signal Descriptions**

Signal	I/O	Description
GPIO[0:23]	I/O	General purpose I/O. Each signal can be set individually to act as input or output, according to application needs.
		<b>State Meaning</b> Asserted/Negated—Defined per application.
		<b>Timing</b> Assertion/Negation—Inputs can be asserted completely asynchronously. Outputs are asynchronous to any externally visible clock

## 21.3 Memory Map/Register Definition

The GPIO has programmable registers that occupy 24 bytes of 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. All addresses used in this chapter are offsets from the address held in IMMRBAR as defined in Chapter 3, “Memory Map.”

Table 21-2 shows the memory map of GPIO.

**Table 21-2. GPIO Register Address Map**

Offset	Register	Access	Reset Value	Section/Page
<b>General Purpose I/O (GPIO)—Block Base Address 0x0_0C00</b>				
0xC00	GPIO direction register (GPDIR)	R/W	0x0000_0000	<a href="#">21.3.1/21-3</a>
0xC04	GPIO open drain register (GPODR)	R/W	0x0000_0000	<a href="#">21.3.2/21-3</a>
0xC08	GPIO data register (GPDAT)	R/W	0x0000_0000	<a href="#">21.3.3/21-4</a>
0xC0C	GPIO interrupt event register (GPIER)	w1c	Undefined	<a href="#">21.3.4/21-4</a>
0xC10	GPIO interrupt mask register (GPIMR)	R/W	0x0000_0000	<a href="#">21.3.5/21-4</a>
0xC14	GPIO external interrupt control register (GPICR)	R/W	0x0000_0000	<a href="#">21.3.6/21-5</a>

### 21.3.1 GPIO Direction Register (GPDIR)

The GPIO direction registers (GPDIR), shown in Figure 21-2, defines the direction of the individual ports.

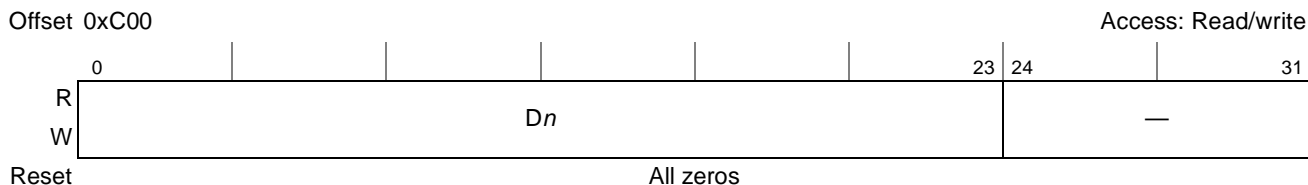


Figure 21-2. GPIO Direction Register (GPDIR)

Table 21-3 defines the bit fields of GPDIR.

Table 21-3. GPDIR Bit Settings

Bits	Name	Description
0–23	<i>Dn</i>	Direction. Indicates whether a signal is used as an input or an output. 0 The corresponding signal is an input. 1 The corresponding signal is an output.
24–31	—	Reserved

### 21.3.2 GPIO Open Drain Register (GPODR)

The GPIO open drain register (GPODR), shown in Figure 21-3, defines the way individual ports drive their output.

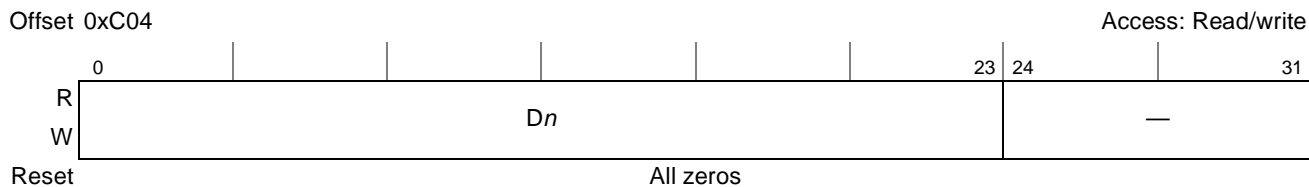


Figure 21-3. GPIO Open Drain Register (GPODR)

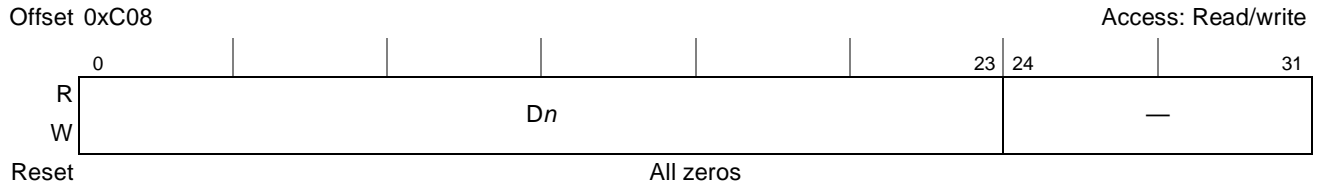
Table 21-4 defines the bit fields of GPODR.

Table 21-4. GPODR Bit Settings

Bits	Name	Description
0–23	<i>Dn</i>	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. 0 The I/O signal is actively driven as an output. 1 The I/O signal is an open-drain driver. As an output, the signal is driven active-low, otherwise it is three-stated.
24–31	—	Reserved

### 21.3.3 GPIO Data Register (GPDAT)

The GPIO data register (GPDAT), shown in [Figure 21-4](#), carries the data in/out for the individual ports.



**Figure 21-4. GPIO Data Register (GPDAT)**

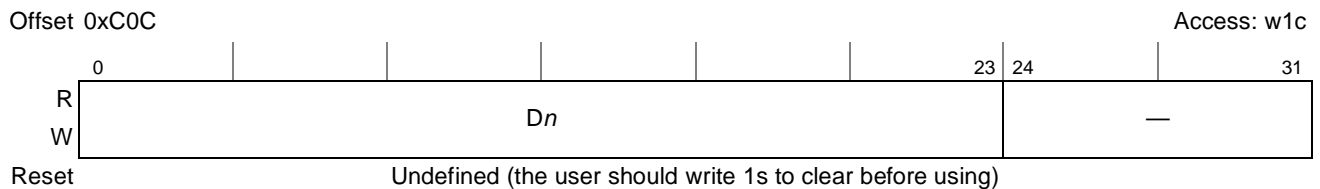
[Table 21-5](#) defines the bit fields of GPDAT.

**Table 21-5. GPnDAT Bit Settings**

Bits	Name	Description
0–23	<i>Dn</i>	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.
24–31	—	Reserved

### 21.3.4 GPIO Interrupt Event Register (GPIER)

The GPIO interrupt event register (GPIER), shown in [Figure 21-5](#), carries information about 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.



**Figure 21-5. GPIO Interrupt Event Register (GPIER)**

[Table 21-6](#) defines the bit fields of GPIER.

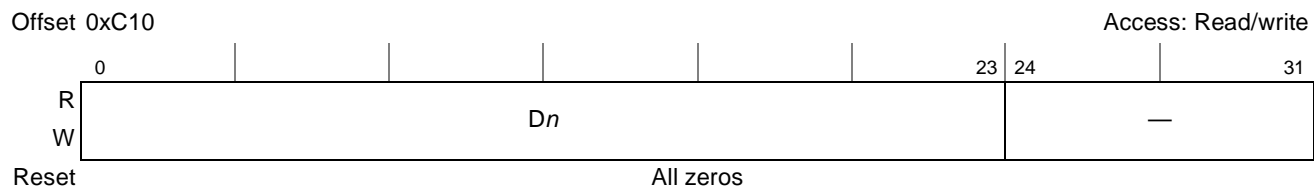
**Table 21-6. GPIER Bit Settings**

Bits	Name	Description
0–23	<i>Dn</i>	Interrupt events. Indicates whether an interrupt event occurred on the corresponding GPIO signal. 0 No interrupt event occurred on the corresponding GPIO signal. 1 Interrupt event occurred on the corresponding GPIO signal.
24–31	—	Reserved

### 21.3.5 GPIO Interrupt Mask Register (GPIMR)

The GPIO interrupt mask register (GPIMR), shown in [Figure 21-6](#), 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.



**Figure 21-6. GPIO Interrupt Mask Register (GPIMR)**

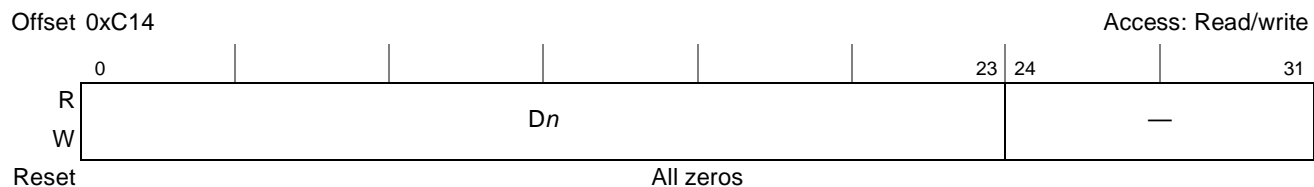
Table 21-7 defines the bit fields of GPIMR.

**Table 21-7. GPIMR Bit Settings**

Bits	Name	Description
0–23	<i>D<sub>n</sub></i>	Interrupt mask. Indicates whether an interrupt event is masked or not masked. 0 The input interrupt signal is masked (disabled). 1 The input interrupt signal is not masked (enabled).
24–31	—	Reserved

### 21.3.6 GPIO Interrupt Control Register (GPICR)

The GPIO interrupt control register (GPICR), shown in Figure 21-7, 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.



**Figure 21-7. GPIO Interrupt Control Register (GPICR)**

Table 21-8 defines the bit fields of GPICR.

**Table 21-8. GPICR Bit Settings**

Bits	Name	Description
0–23	<i>D<sub>n</sub></i>	Edge detection mode. The corresponding port line asserts an interrupt request according to the following: 0 Any change on the state of the port generates an interrupt request. 1 High-to-low change on the port generates an interrupt request.
24–31	—	Reserved





# Appendix A

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

### A.1 Local Access Windows

Table A-1. Local Access Register Memory Map

Local Access—Block Base Address 0x0_000				
Local Memory Offset (Hex)	Register	Access	Reset	Section/Page
0x000	Internal memory map base address register (IMMRBAR)	R/W	0xFF40_0000	5.1.4.1/5-5
0x004	Reserved	—	—	—
0x008	Alternate configuration base address register (ALTCBAR)	R/W	0x0000_0000	5.1.4.2/5-7
0x00C–0x01C	Reserved	—	—	—
0x020	eLBC local access window 0 base address register (LBLAWBAR0)	R/W	0x0000_0000 <sup>1</sup>	5.1.4.3/5-7
0x024	eLBC local access window 0 attribute register (LBLAWAR0)	R/W	0x0000_0000 <sup>2</sup>	5.1.4.4/5-8
0x028	eLBC local access window 1 base address register (LBLAWBAR1)	R/W	0x0000_0000	5.1.4.3/5-7
0x02C	eLBC local access window 1 attribute register (LBLAWAR1)	R/W	0x0000_0000	5.1.4.4/5-8
0x030	eLBC local access window 2 base address register (LBLAWBAR2)	R/W	0x0000_0000	5.1.4.3/5-7
0x034	eLBC local access window 2 attribute register (LBLAWAR2)	R/W	0x0000_0000	5.1.4.4/5-8
0x038	eLBC local access window 3 base address register (LBLAWBAR3)	R/W	0x0000_0000	5.1.4.3/5-7
0x03C	eLBC local access window 3 attribute register (LBLAWAR3)	R/W	0x0000_0000	5.1.4.4/5-8
0x040–0x063C	Reserved	—	—	—
0x064	Reserved	—	—	—
0x068–0x07C	Reserved	—	—	—
0x080	PCI Express local access window base address register (PCIEXP1LAWBAR)	R/W	0x0000_0000	5.1.4.5/5-9
0x084	PCI Express local access window attribute register (PCIEXP1LAWAR)	R/W	0x0000_0000	5.1.4.6/5-10
0x088–0x09C	Reserved	—	—	—
0x0A0	DDR2 local access window 0 base address register (DDRLAWBAR0)	R/W	0x0000_0000 <sup>3</sup>	5.1.4.7/5-11
0x0A4	DDR2 local access window 0 attribute register (DDRLAWAR0)	R/W	0x0000_0000 <sup>4</sup>	5.1.4.8/5-12

Table A-1. Local Access Register Memory Map (continued)

Local Access—Block Base Address 0x0_000				
Local Memory Offset (Hex)	Register	Access	Reset	Section/Page
0x0A8	DDR2 local access window 1 base address register (DDRLAWBAR1)	R/W	0x0000_0000	5.1.4.7/5-11
0x0AC	DDR2 local access window 1 attribute register (DDRLAWAR1)	R/W	0x0000_0000	5.1.4.8/5-12
0x0B0–0x0FC	Reserved	—	—	—

<sup>1</sup> Depends on reset configuration word high values. See Section 5.1.4.3.1, “LBLAWBAR0[BASE\_ADDR] Reset Value,” for details.

<sup>2</sup> Depends on reset configuration word high values. See Section 5.1.4.4.1, “LBLAWAR0[EN] and LBLAWAR0[SIZE] Reset Value,” for details.

<sup>3</sup> Depends on reset configuration word high values. See Section 5.1.4.7.1, “DDRLAWBAR0[BASE\_ADDR] Reset Value,” for details.

<sup>4</sup> Depends on reset configuration word high values. See Section 5.1.4.8.1, “DDRLAWAR0[EN] and DDRLAWAR0[SIZE] Reset Value,” for details.

## A.2 System Configuration Registers

Table A-2. System Configuration Registers

Local Memory Offset (Hex)	Register	Access	Reset	Section/Page
System Configuration—Block Base Address 0x0_000				
0x100	System general purpose register low (SGPRL)	R/W	0x0000_0000	5.2.2.1/5-16
0x104	System general purpose register high (SGPRH)	R/W	0x0000_0000	5.2.2.2/5-16
0x108	System part and revision ID register (SPRIDR)	R	0x8101_0010	5.2.2.3/5-17
0x10C	Reserved	—	—	—
0x110	System priority configuration register (SPCR)	R/W	0x0000_0000	5.2.2.4/5-17
0x114	System I/O configuration register low (SICRL)	R/W	0x0000_0000 <sup>1</sup>	5.2.2.5/5-19
0x118	System I/O configuration register high (SICRH)	R/W	0x0014_5000 <sup>2</sup>	5.2.2.6/5-22
0x11C–0x124	Reserved	—	—	—
0x128	DDR control driver register (DDRCDR)	R/W	0x7304_0001	5.2.2.9/5-26
0x12C	DDR debug status register (DDRDSR)	R	0x3300_0000	5.2.2.10/5-27
0x140	PCI Express control register 1 (PECR1)	R/W	0x0000_0000	5.2.2.11/5-28
0x144	eSDHC Control Register (SDHCCR)	R/W	0x0000_0000	5.2.2.12/5-29
0x148	RTC Control Register (RTCCR)	R/W	0x0000_0000	5.2.2.13/5-31
0x160–0x1FC	Reserved	—	—	—

<sup>1</sup> Bit #25 depends on the RCW.

<sup>2</sup> Bit #30 depend on RCW.

## A.3 Watchdog Timer (WDT)

Table A-3. Watchdog Timer (WDT) Registers

Watchdog Timer (WDT)—Block Base Address 0x0_0200				
Offset	Register	Access	Reset	Section/Page
0x000–0x003	Reserved	—	—	—
0x004	System watchdog control register (SWCRR)	R/W	0xFFFF_0003 or 0xFFFF_0007 <sup>1</sup>	5.3.4.1/5-33
0x008	System watchdog count register (SWCNR)	R	0x0000_FFFF	5.3.4.2/5-34
0x00C–0x00D	Reserved	—	—	—
0x00E	System watchdog service register (SWSRR)	R/W	0x0000	5.3.4.3/5-35

<sup>1</sup> SWCRR[SWEN] reset value directly depends on RCWHR[SWEN] (reset configuration word high).

## A.4 Real Time Clock (RTC)

Table A-4. Real Time Clock (RTC) Registers

Offset	Register	Access	Reset Value <sup>1</sup>	Section/Page
Real Time Clock (RTC)—Block Base Address 0x0_0300				
0x000	Real time counter control register (RTCNR)	R/W	0x0000_0000	5.4.6.1/55-41
0x004	Real time counter load register (RTLDR)	R/W	0x0000_0000	5.4.6.2/55-42
0x008	Real time counter prescale register (RTPSR)	R/W	0x0000_0000	5.4.6.3/55-42
0x00C	Real time counter register (RTCTR)	R	0x0000_0000	5.4.6.4/55-43
0x010	Real time counter event register (RTEVR)	w1c	0x0000_0000	5.4.6.5/55-43
0x014	Real time counter alarm register (RTALR)	R/W	0xFFFF_FFFF	5.4.6.6/55-44
0x018–0x01F	Reserved	—	—	

<sup>1</sup> It refers to the case where software writes to a specific RTC register, RTCCR. For more information, see [Section 5.4.8, “RTC Reset Sequence.”](#)

## A.5 Periodic Interval Timer (PIT)

Table A-5. Periodic Interval Timer (PIT) Registers

Periodic Interval Timer (PIT)—Block Base Address 0x0_0400				
Offset	Register	Access	Reset	Section/Page
0x000	Periodic interval timer control register (PTCNR)	R/W	0x0000_0000	5.5.5.1/5-48
0x004	Periodic interval timer load register (PTLDR)	R/W	0x0000_0000	5.5.5.2/5-49
0x008	Periodic interval timer prescale register (PTPSR)	R/W	0x0000_0000	5.5.5.3/5-49
0x00C	Periodic interval timer counter register (PTCTR)	R	0x0000_0000	5.5.5.4/5-50

Table A-5. Periodic Interval Timer (PIT) Registers (continued)

Periodic Interval Timer (PIT)—Block Base Address 0x0_0400				
Offset	Register	Access	Reset	Section/Page
0x010	Periodic interval timer event register (PTEVR)	w1c	0x0000_0000	5.5.5.5/5-50
0x014–0x01F	Reserved	—	—	—

## A.6 General Purpose (Global) Timers (GTMs)

Table A-6. General Purpose (Global) Timers (GTMs) Registers

Offset	Register	Access	Reset Value	Section/Page
<b>General Purpose (Global) Timer Module 1—Block Base Address 0x0_0500</b>				
0x000	Timer 1 and 2 global timers configuration register (GTCFR1)	R/W	0x0000	5.6.5.1/5-58
0x001–0x003	Reserved	—	—	—
0x004	Timer 3 and 4 global timers configuration register (GTCFR2)	R/W	0x0000	5.6.5.1/5-58
0x005–0x00F	Reserved	—	—	—
0x010	Timer 1 global timers mode register (GTMDR1)	R/W	0x0000	5.6.5.2/5-61
0x012	Timer 2 global timers mode register (GTMDR2)			
0x014	Timer 1 global timers reference register (GTRFR1)	R/W	0xFFFF	5.6.5.3/5-62
0x016	Timer 2 global timers reference register (GTRFR2)			
0x018	Timer 1 global timers capture register (GTCPR1)	R/W	0x0000	5.6.5.4/5-62
0x01A	Timer 2 global timers capture register (GTCPR2)			
0x01C	Timer 1 global timers counter register (GTCNR1)	R/W	0x0000	5.6.5.5/5-63
0x01E	Timer 2 global timers counter register (GTCNR2)			
0x020	Timer 3 global timers mode register (GTMDR3)	R/W	0x0000	5.6.5.2/5-61
0x022	Timer 4 global timers mode register (GTMDR4)			
0x024	Timer 3 global timers reference register (GTRFR3)	R/W	0xFFFF	5.6.5.3/5-62
0x026	Timer 4 global timers reference register (GTRFR4)			
0x028	Timer 3 global timers capture register (GTCPR3)	R	0x0000	5.6.5.4/5-62
0x02A	Timer 4 global timers capture register (GTCPR4)			
0x02C	Timer 3 global timers counter register (GTCNR3)	R/W	0x0000	5.6.5.5/5-63
0x02E	Timer 4 global timers counter register (GTCNR4)			
0x030	Timer 1 global timers event register (GTEVR1)	w1c	0x0000	5.6.5.6/5-63
0x032	Timer 2 global timers event register (GTEVR2)			
0x034	Timer 3 global timers event register (GTEVR3)			
0x036	Timer 4 global timers event register (GTEVR4)			

**Table A-6. General Purpose (Global) Timers (GTMs) Registers (continued)**

Offset	Register	Access	Reset Value	Section/Page
0x038	Timer 1 global timers prescale register (GTPSR1)	R/W	0x0003	5.6.5.7/5-64
0x03A	Timer 2 global timers prescale register (GTPSR2)			
0x03C	Timer 3 global timers prescale register (GTPSR3)			
0x03E	Timer 4 global timers prescale register (GTPSR4)			

## A.7 Integrated Programmable Interrupt Controller (IPIC)

**Table A-7. IPIC Registers**

Integrated Programmable Interrupt Controller—Block Base Address 0x0_0700				
Offset	Register	Access	Reset Value	Section/Page
0x000	System global interrupt configuration register (SICFR)	R/W	0x0000_0000	8.5.1/8-7
0x004	System regular interrupt vector register (SIVCR)	R	0x0000_0000	8.5.2/8-8
0x008	System internal interrupt pending register (SIPNR_H)	R	0x0000_0000	8.5.3/8-11
0x00C	System internal interrupt pending register (SIPNR_L)	R	0x0000_0000	8.5.3/8-11
0x010	System internal interrupt group A priority register (SIPRR_A)	R/W	0x0530_9770	8.5.4/8-13
0x014	System internal interrupt group B priority register (SIPRR_B)	R/W	0x0530_9770	8.5.5/8-14
0x018	System internal interrupt group C priority register (SIPRR_C)	R/W	0x0530_9770	8.5.6/8-15
0x01C	System internal interrupt group D priority register (SIPRR_D)	R/W	0x0530_9770	8.5.7/8-16
0x020	System internal interrupt mask register (SIMSR_H)	R/W	0x0000_0000	8.5.8/8-16
0x024	System internal interrupt mask register (SIMSR_L)	R/W	0x0000_0000	8.5.8/8-16
0x028	System internal interrupt control register (SICNR)	R/W	0x0000_0000	8.5.9/8-18
0x02C	System external interrupt pending register (SEPNR)	R/W	Special	8.5.10/8-20
0x030	System mixed interrupt group A priority register (SMPRR_A)	R/W	0x0530_9770	8.5.11/8-20
0x034	System mixed interrupt group B priority register (SMPRR_B)	R/W	0x0530_9770	8.5.12/8-21
0x038	System external interrupt mask register (SEMSR)	R/W	0x0000_0000	8.5.13/8-22
0x03C	System external interrupt control register (SECNR)	R/W	0x0000_0000	8.5.14/8-23
0x040	System error status register (SERSR)	R/W	0x0000_0000	8.5.15/8-24
0x044	System error mask register (SERMR)	R/W		8.5.16/8-25
0x048	System error control register (SERCR)	R/W	0x0000_0000	8.5.17/8-26
0x04C	System external interrupt polarity control register (SEPCR)	R/W	0x0000_0000	8.5.18/8-26
0x04F	Reserved	—	—	—
0x050	System internal interrupt force register (SIFCR_H)	R/W	0x0000_0000	8.5.19/8-27
0x054	System internal interrupt force register (SIFCR_L)	R/W	0x0000_0000	8.5.19/8-27

Table A-7. IPIC Registers (continued)

Integrated Programmable Interrupt Controller—Block Base Address 0x0_0700				
Offset	Register	Access	Reset Value	Section/Page
0x058	System external interrupt force register (SEFCR)	R/W	0x0000_0000	8.5.20/8-28
0x05C	System error force register (SERFR)	R/W	0x0000_0000	8.5.21/8-29
0x060	System critical interrupt vector register (SCVCR)	R	0x0000_0000	8.5.22/8-29
0x064	System management interrupt vector register (SMVCR)	R	0x0000_0000	8.5.23/8-30
0x068–0x0BF	Reserved	—	—	—

## A.8 System Arbiter

Table A-8. System Arbiter Registers

System Arbiter—Block Base Address 0x0_0800				
Offset	Register	Access	Reset	Section/Page
0x00	Arbiter configuration register (ACR)	R/W	0x0000_0000/ 0x0010_0000 <sup>1</sup>	6.2.1/6-3
0x04	Arbiter timers register (ATR)	R/W	FFFF_FFFF	6.2.2/6-4
0x0C	Arbiter event register (AER)	w1c	0x0000_0000	6.2.3/6-5
0x10	Arbiter interrupt definition register (AIDR)	R/W	0x0000_0000	6.2.4/6-6
0x14	Arbiter mask register (AMR)	R/W	0x0000_0000	6.2.5/6-7
0x18	Arbiter event attributes register (AEATR)	R	0x0000_0000 <sup>2</sup>	6.2.6/6-8
0x1C	Arbiter event address register (AEADR)	R	0x0000_0000 <sup>2</sup>	6.2.7/6-9
0x20	Arbiter event response register (AERR)	R/W	0x0000_0000	6.2.8/6-10

<sup>1</sup> Reset value is determined from the core PLL configuration of the reset configuration word. See Chapter 4, “Reset, Clocking, and Initialization,” for details.

<sup>2</sup> The registers AEATR and AEADR are affected only by the assertion of  $\overline{\text{PORESET}}$ .

## A.9 Reset Configuration

Table A-9. Reset Configuration Registers

Reset Configuration—Block Base Address 0x0_0900				
Offset	Register	Access	Reset	Section/Page
0x000	Reset configuration word low register (RCWLR)	R	0x0000_0000	4.5.1.1/4-25
0x004	Reset configuration word high register (RCWHR)	R	0x0000_0000	4.5.1.2/4-26
0x008– 0x00C	Reserved, should be cleared	—	—	—
0x010	Reset status register (RSR)	R/W	0x0000_0000	4.5.1.3/4-26

Table A-9. Reset Configuration Registers (continued)

Reset Configuration—Block Base Address 0x0_0900				
Offset	Register	Access	Reset	Section/Page
0x014	Reset mode register (RMR)	R/W	0x0000_0000	<a href="#">4.5.1.4/4-27</a>
0x018	Reset protection register (RPR)	R/W	0x0000_0000	<a href="#">4.5.1.5/4-28</a>
0x01C	Reset control register (RCR)	R/W	0x0000_0000	<a href="#">4.5.1.6/4-28</a>
0x020	Reset control enable register (RCER)	R/W	0x0000_0000	<a href="#">4.5.1.7/4-29</a>
0x024	Reserved.	—	—	—
0x028–0x0FC	Reserved, should be cleared	—	—	—

## A.10 Clock Configuration

Table A-10. Clock Configuration Registers

Clock Configuration—Block Base Address 0x0_0A00				
Offset	Register	Access	Reset	Section/Page
0x000	System PLL mode register (SPMR)	R	0xn <sub>nnn</sub> _n <sub>nnn</sub>	<a href="#">4.5.2.1/4-30</a>
0x004	Output clock control register (OCCR)	R/W	0x0000_E080	<a href="#">4.5.2.2/4-31</a>
0x008	System clock control register (SCCR)	R/W	0x5550_0010	<a href="#">4.5.2.3/4-32</a>
0x00C–0x0FC	Reserved, should be cleared	—	—	—

## A.11 Power Management Controller (PMC)

Table A-11. Power Management Controller (PMC) Registers

Power Management Controller—Block Base Address 0x0_0B00				
Offset	Register	Access	Reset	Section/Page
0x000	Power management controller configuration register (PMCCR)	R/W	0x0000_0000	<a href="#">5.7.2.1/5-69</a>
0x004–0x0FC	Reserved	—	—	—

## A.12 General Purpose I/O (GPIO)

Table A-12. General Purpose I/O (GPIO) Registers

General Purpose I/O (GPIO)—Block Base Address 0x0_0C00				
Offset	Register	Access	Reset	Section/Page
0x000	GPIO direction register (GPDIR)	R/W	0x0000_0000	<a href="#">21.3.1/21-3</a>
0x004	GPIO open drain register (GPODR)	R/W	0x0000_0000	<a href="#">21.3.2/21-3</a>

Table A-12. General Purpose I/O (GPIO) Registers (continued)

General Purpose I/O (GPIO)—Block Base Address 0x0_0C00				
Offset	Register	Access	Reset	Section/Page
0x008	GPIO data register (GPDAT)	R/W	0x0000_0000	21.3.3/21-4
0x00C	GPIO interrupt event register (GPIER)	w1c	Undefined	21.3.4/21-4
0x010	GPIO interrupt mask register (GPIMR)	R/W	0x0000_0000	21.3.5/21-4
0x014	GPIO external interrupt control register (GPICR)	R/W	0x0000_0000	21.3.6/21-5

## A.13 DDR Memory Controller

Table A-13. 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	9.4.1.1/9-10
0x008	CS1_BNDS—Chip select 1 memory bounds	R/W	0x0000_0000	9.4.1.1/9-10
0x080	CS0_CONFIG—Chip select 0 configuration	R/W	0x0000_0000	9.4.1.2/9-11
0x084	CS1_CONFIG—Chip select 1 configuration	R/W	0x0000_0000	9.4.1.2/9-11
0x100	TIMING_CFG_3—DDR SDRAM timing configuration 3	R/W	0x0000_0000	9.4.1.3/9-13
0x104	TIMING_CFG_0—DDR SDRAM timing configuration 0	R/W	0x0011_0105	9.4.1.4/9-14
0x108	TIMING_CFG_1—DDR SDRAM timing configuration 1	R/W	0x0000_0000	9.4.1.5/9-16
0x10C	TIMING_CFG_2—DDR SDRAM timing configuration 2	R/W	0x0000_0000	9.4.1.6/9-18
0x110	DDR_SDRAM_CFG—DDR SDRAM control configuration	R/W	0x0200_0000	9.4.1.7/9-19
0x114	DDR_SDRAM_CFG_2—DDR SDRAM control configuration 2	R/W	0x0000_0000	9.4.1.8/9-22
0x118	DDR_SDRAM_MODE—DDR SDRAM mode configuration	R/W	0x0000_0000	9.4.1.9/9-23
0x11C	DDR_SDRAM_MODE_2—DDR SDRAM mode configuration 2	R/W	0x0000_0000	9.4.1.10/9-24
0x120	DDR_SDRAM_MD_CNTL—DDR SDRAM mode control	R/W	0x0000_0000	9.4.1.11/9-25
0x124	DDR_SDRAM_INTERVAL—DDR SDRAM interval configuration	R/W	0x0000_0000	9.4.1.12/9-27
0x128	DDR_DATA_INIT—DDR SDRAM data initialization	R/W	0x0000_0000	9.4.1.13/9-28
0x130	DDR_SDRAM_CLK_CNTL—DDR SDRAM clock control	R/W	0x0200_0000	9.4.1.14/9-28
0x140–0x144	Reserved	—	—	—
0x148	DDR_INIT_ADDR—DDR training initialization address	R/W	0x0000_0000	9.4.1.15/9-29
0x150–0xBF4	Reserved	—	—	—
0xBF8	DDR_IP_REV1—DDR IP block revision 1	R	0xn <sup>1</sup> nnn <sup>1</sup> _nnn <sup>1</sup>	9.4.1.16/9-29
0xBFC	DDR_IP_REV2—DDR IP block revision 2	R	0x00nn <sup>1</sup> _00nn <sup>1</sup>	9.4.1.17/9-30
0xE00	DATA_ERR_INJECT_HI—Memory data path error injection mask high	R/W	0x0000_0000	9.4.1.18/9-30
0xE04	DATA_ERR_INJECT_LO—Memory data path error injection mask low	R/W	0x0000_0000	9.4.1.19/9-31
0xE08	ERR_INJECT—Memory data path error injection mask ECC	R/W	0x0000_0000	9.4.1.20/9-31



Table A-13. DDR Memory Controller Registers (continued)

DDR Memory Controller—Block Base Address 0x0_2000				
Offset	Register	Access	Reset	Section/Page
0xE20	CAPTURE_DATA_HI—Memory data path read capture high	R/W	0x0000_0000	9.4.1.21/9-32
0xE24	CAPTURE_DATA_LO—Memory data path read capture low	R/W	0x0000_0000	9.4.1.22/9-32
0xE28	CAPTURE_ECC—Memory data path read capture ECC	R/W	0x0000_0000	9.4.1.23/9-33
0xE40	ERR_DETECT—Memory error detect	w1c	0x0000_0000	9.4.1.24/9-33
0xE44	ERR_DISABLE—Memory error disable	R/W	0x0000_0000	9.4.1.25/9-34
0xE48	ERR_INT_EN—Memory error interrupt enable	R/W	0x0000_0000	9.4.1.26/9-35
0xE4C	CAPTURE_ATTRIBUTES—Memory error attributes capture	R/W	0x0000_0000	9.4.1.27/9-36
0xE50	CAPTURE_ADDRESS—Memory error address capture	R/W	0x0000_0000	9.4.1.28/9-37
0xE54	Reserved	—	—	—
0xE58	ERR_SBE—Single-Bit ECC memory error management	R/W	0x0000_0000	9.4.1.29/9-37

<sup>1</sup> Implementation-dependent reset values are listed in specified section/page.

## A.14 I<sup>2</sup>C Controller

Table A-14. I<sup>2</sup>C Controller Registers

I <sup>2</sup> C Controller 1 —Block Base Address 0x0_3000 I <sup>2</sup> C Controller 2 —Block Base Address 0x0_3100				
Offset	Register	Access	Reset	Section/Page
0x000	I2CADR—I <sup>2</sup> C address register	R/W	0x0000	17.3.1.1/17-5
0x004	I2CFDR—I <sup>2</sup> C frequency divider register	R/W	0x0000	17.3.1.2/17-5
0x008	I2CCR—I <sup>2</sup> C control register	R/W	0x0000	17.3.1.3/17-6
0x00C	I2CSR—I <sup>2</sup> C status register	R/W	0x0081	17.3.1.4/17-8
0x010	I2CDR—I <sup>2</sup> C data register	R/W	0x0000	17.3.1.5/17-9
0x014	I2CDFSRR—I <sup>2</sup> C digital filter sampling rate register	R/W	0x0010	17.3.1.6/17-10
0x01C–0x1FF	Reserved	—	—	—

## A.15 DUART

Table A-15. DUART Registers

UART 1—Block Base Address 0x0_4000 UART 2—Block Base Address 0x0_4100				
Offset	Register	Access	Reset	Section/Page
0x500	URBR—ULCR[DLAB] = 0 UART1 receiver buffer register	R	0x0000	18.3.1.1/18-5
	UTHR—ULCR[DLAB] = 0 UART1 transmitter holding register	W	0x0000	18.3.1.2/18-5
	UDLB—ULCR[DLAB] = 1 UART1 divisor least significant byte register	R/W	0x0000	18.3.1.3/18-6

Table A-15. DUART Registers (continued)

UART 1—Block Base Address 0x0_4000 UART 2—Block Base Address 0x0_4100				
Offset	Register	Access	Reset	Section/Page
0x501	UIER—ULCR[DLAB] = 0 UART1 interrupt enable register	R/W	0x0000	18.3.1.4/18-7
	UDMB—ULCR[DLAB] = 1 UART1 divisor most significant byte register	R/W	0x0000	18.3.1.3/18-6
0x502	UIIR—ULCR[DLAB] = 0 UART1 interrupt ID register	R	0x0001	18.3.1.5/18-8
	UFCR—ULCR[DLAB] = 0 UART1 FIFO control register	W	0x0000	18.3.1.6/18-9
	UAFR—ULCR[DLAB] = 1 UART1 alternate function register	R/W	0x0000	18.3.1.11/18-14
0x503	ULCR—ULCR[DLAB] = x UART1 line control register	R/W	0x0000	18.3.1.7/18-10
0x504	UMCR—ULCR[DLAB] = x UART1 MODEM control register	R/W	0x0000	18.3.1.8/18-12
0x505	ULSR—ULCR[DLAB] = x UART1 line status register	R	0x0060	18.3.1.9/18-13
0x506	Reserved	—	—	—
0x507	USCR—ULCR[DLAB] = x UART1 scratch register	R/W	0x0000	18.3.1.10/18-14
0x510	UDSR—ULCR[DLAB] = x UART1 DMA status register	R	0x0001	18.3.1.12/18-15
0x600	URBR—ULCR[DLAB] = 0 UART2 receiver buffer register	R	0x0000	18.3.1.1/18-5
	UTHR—ULCR[DLAB] = 0 UART2 transmitter holding register	W	0x0000	18.3.1.2/18-5
	UDLB—ULCR[DLAB] = 1 UART2 divisor least significant byte register	R/W	0x0000	18.3.1.3/18-6
0x601	UIER—ULCR[DLAB] = 0 UART2 interrupt enable register	R/W	0x0000	18.3.1.4/18-7
	UDMB—ULCR[DLAB] = 1 UART2 divisor most significant byte register	R/W	0x0000	18.3.1.3/18-6
0x602	UIIR—ULCR[DLAB] = 0 UART2 interrupt ID register	R	0x0001	18.3.1.5/18-8
	UFCR—ULCR[DLAB] = 0 UART2 FIFO control register	W	0x0000	18.3.1.6/18-9
	UAFR—ULCR[DLAB] = 1 UART2 alternate function register	R/W	0x0000	18.3.1.11/18-14
0x603	ULCR—ULCR[DLAB] = x UART2 line control register	R/W	0x0000	18.3.1.7/18-10
0x604	UMCR—ULCR[DLAB] = x UART2 MODEM control register	R/W	0x0000	18.3.1.8/18-12
0x605	ULSR—ULCR[DLAB] = x UART2 line status register	R	0x0060	18.3.1.9/18-13
0x606	Reserved	—	—	—
0x607	USCR—ULCR[DLAB] = x UART2 scratch register	R/W	0x0000	18.3.1.10/18-14
0x610	UDSR—ULCR[DLAB] = x UART2 DMA status register	R	0x0001	18.3.1.12/18-15

## A.16 Enhanced Local Bus Controller (eLBC)

Table A-16. 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	<a href="#">10.3.1.1/10-9</a>
0x008	BR1—Base register 1	R/W	0x0000_0000	<a href="#">10.3.1.1/10-9</a>
0x010	BR2—Base register 2	R/W	0x0000_0000	<a href="#">10.3.1.1/10-9</a>
0x018	BR3—Base register 3	R/W	0x0000_0000	<a href="#">10.3.1.1/10-9</a>
0x020–0x038	Reserved	—	—	—
0x004	OR0—Options register 0	R/W	0x0000_0FF7	<a href="#">10.3.1.2/10-10</a>
0x00C	OR1—Options register 1	R/W	0x0000_0000	<a href="#">10.3.1.2/10-10</a>
0x014	OR2—Options register 2	R/W	0x0000_0000	<a href="#">10.3.1.2/10-10</a>
0x01C	OR3—Options register 3	R/W	0x0000_0000	<a href="#">10.3.1.2/10-10</a>
0x024–0x064	Reserved	—	—	—
0x068	MAR—UPM address register	R/W	0x0000_0000	<a href="#">10.3.1.3/10-18</a>
0x06C	Reserved	—	—	—
0x070	MAMR—UPMA mode register	R/W	0x0000_0000	<a href="#">10.3.1.4/10-19</a>
0x074	MBMR—UPMB mode register	R/W	0x0000_0000	<a href="#">10.3.1.4/10-19</a>
0x078	MCMR—UPMC mode register	R/W	0x0000_0000	<a href="#">10.3.1.4/10-19</a>
0x07C–0x080	Reserved	—	—	—
0x084	MRTPR—Memory refresh timer prescaler register	R/W	0x0000_0000	<a href="#">10.3.1.5/10-21</a>
0x088	MDR—UPM/FCM data register	R/W	0x0000_0000	<a href="#">10.3.1.6/10-21</a>
0x08C	Reserved	—	—	—
0x090	LSOR—Special operation initiation register	R/W	0x0000_0000	<a href="#">10.3.1.7/10-22</a>
0x094–0x09C	Reserved	—	—	—
0x0A0	LURT—UPM refresh timer	R/W	0x0000_0000	<a href="#">10.3.1.4/10-19</a>
0x0A4–0x0AC	Reserved	—	—	—
0x0B0	LTESR—Transfer error status register	w1c	0x0000_0000	<a href="#">10.3.1.9/10-24</a>
0x0B4	LTEDR—Transfer error disable register	R/W	0x0000_0000	<a href="#">10.3.1.10/10-26</a>
0x0B8	LTEIR—Transfer error interrupt register	R/W	0x0000_0000	<a href="#">10.3.1.11/10-27</a>
0x0BC	LTEATR—Transfer error attributes register	R/W	0x0000_0000	<a href="#">10.3.1.12/10-28</a>
0x0C0	LTEAR—Transfer error address register	R/W	0x0000_0000	<a href="#">10.3.1.13/10-29</a>

Table A-16. Enhanced Local Bus Controller Registers (continued)

Enhanced Local Bus Controller—Block Base Address 0x0_5000				
Offset	Register	Access	Reset	Section/Page
0x0C4	LTECCR—Transfer error ECC register	w1c	0x0000_0000	<a href="#">10.3.1.14/10-29</a>
0x0C8–0x0CC	Reserved	—	—	—
0x0D0	LBCR—Configuration register	R/W	0x0004_0000	<a href="#">10.3.1.15/10-30</a>
0x0D4	LCRR—Clock ratio register	R/W	0x8000_0008	<a href="#">10.3.1.16/10-31</a>
0x0D8–0x0DC	Reserved	—	—	—
0x0E0	FMR—Flash mode register	R/W	0x0000_0n00	<a href="#">10.3.1.17/10-32</a>
0x0E4	FIR—Flash instruction register	R/W	0x0000_0000	<a href="#">10.3.1.18/10-34</a>
0x0E8	FCR—Flash command register	R/W	0x0000_0000	<a href="#">10.3.1.19/10-35</a>
0x0EC	FBAR—Flash block address register	R/W	0x0000_0000	<a href="#">10.3.1.20/10-36</a>
0x0F0	FPAR—Flash page address register	R/W	0x0000_0000	<a href="#">10.3.1.21/10-36</a>
0x0F4	FBCR—Flash byte count register	R/W	0x0000_0000	<a href="#">10.3.1.22/10-38</a>
0x0F8–0x0FC	Reserved	—	—	—
0x100	FECC0—Flash ECC block 0 register	R	0x0000_0000	<a href="#">10.3.1.23/10-38</a>
0x104	FECC1—Flash ECC block 1 register	R	0x0000_0000	<a href="#">10.3.1.23/10-38</a>
0x108	FECC2—Flash ECC block 2 register	R	0x0000_0000	<a href="#">10.3.1.23/10-38</a>
0x10C	FECC3—Flash ECC block 3 register	R	0x0000_0000	<a href="#">10.3.1.23/10-38</a>

## A.17 Serial Peripheral Interface (SPI)

Table A-17. Serial Peripheral Interface (SPI) Registers

Serial Peripheral Interface (SPI)—Block Base Address 0x0_7000				
Offset	Register	Access	Reset	Section/Page
0x000–0x01F	Reserved	—	—	—
0x020	SPI mode register (SPMODE)	R/W	0x0000_0000	<a href="#">19.3.1.1/19-8</a>
0x024	SPI event register (SPIE)	Mixed	0x0000_0000	<a href="#">19.3.1.2/19-10</a>
0x028	SPI mask register (SPIM)	R/W	0x0000_0000	<a href="#">19.3.1.3/19-11</a>
0x02C	SPI command register (SPCOM)	W	0x0000_0000	<a href="#">19.3.1.4/19-13</a>
0x030	SPI transmit register (SPITD)	W	0x0000_0000	<a href="#">19.3.1.5/19-13</a>
0x034	SPI receive register (SPIRD)	R	0xFFFF_FFFF	<a href="#">19.3.1.6/19-14</a>
0x038–0xFFFF	Reserved	—	—	—

## A.18 DMA Controller

Table A-18. DMA Controller Registers

Block Base Address: 0x2_C000				
Offset	Register	Access	Reset	Section/Page
0x000	DMACR—DMA Control Register	R/W	0x0000_E400	<a href="#">12.2.1/12-3</a>
0x004	DMAES—DMA Error Status Register	R	0x0000_0000	<a href="#">12.3/12-6</a>
0x008– 0x010	Reserved	—	—	—
0x014	DMAEEI—DMA enable error interrupt register	R/W	0x0000_0000	<a href="#">12.3.1/12-8</a>
0x018– 0x019	Reserved	—	—	—
0x01A	DMASEEI—DMA Set Enable Error Interrupt	R/W	0x0000	<a href="#">12.3.2/12-9</a>
0x01B	DMACEEI—DMA Clear Enable Error Interrupt	R/W	0x0000	<a href="#">12.3.3/12-9</a>
0x01C	DMACINT—DMA Clear Interrupt Request	R/W	0x0000	<a href="#">12.3.4/12-10</a>
0x01D	DMACERR—DMA Clear Error	R/W	0x0000	<a href="#">12.3.5/12-11</a>
0x01E	DMASSRT—DMA Set START Bit	R/W	0x0000	<a href="#">12.3.6/12-11</a>
0x01F	DMACDNE—DMA Clear DONE Status Bit	R/W	0x0000	<a href="#">12.3.7/12-12</a>
0x020	Reserved	—	—	—
0x024	DMAINT—DMA interrupt request register	w1c	0x0000_0000	<a href="#">12.3.8/12-12</a>
0x028	Reserved	—	—	—
0x02C	DMAERR—DMA error register	w1c	0x0000_0000	<a href="#">12.3.9/12-13</a>
0x030– 0x034	Reserved	—	—	—
0x038	DMAGPOR—DMA general purpose output register	R/W	0x0000_0000	<a href="#">12.3.10/12-14</a>
0x03C– 0x0FC	Reserved	—	—	—
0x100– 0x13C	DCHPRI <sub>n</sub> —DMA Channel <i>n</i> Priority Register	R/W	0x0000_ <i>nnnn</i>	<a href="#">12.3.11/12-15</a>
0x140– 0xFFC	Reserved	—	—	—

## A.19 PCI Express Controller

Table A-20. PCI Express Controller Registers

PCI Express—Block Base Address 0x0_9000				
Offset	Register	Access	Reset	Section/Page
<b>PCI Express Controller Registers</b>				
<b>PCI Express Core Configuration Header Registers</b>				
0x000	PCI Express Vendor ID Register	R	0x1957	<a href="#">14.4.1.1/14-15</a>
0x002	PCI Express Device ID Register	R	Device-specific	<a href="#">14.4.1.2/14-15</a>
0x004	PCI Express Command Register	Mixed	0x0000	<a href="#">14.4.1.3/14-15</a>
0x006	PCI Express Status Register	Mixed	0x0010	<a href="#">14.4.1.4/14-17</a>
0x008	PCI Express Revision ID Register	R	Revision-specific	<a href="#">14.4.1.5/14-18</a>
0x009	PCI Express Class Code Register	Mixed	0x0B20	<a href="#">14.4.1.6/14-18</a>
0x00C	PCI Express Cache Line Size Register	R/W	0x00	<a href="#">14.4.1.7/14-19</a>
0x00D	PCI Express Latency Timer Register	R	0x00	<a href="#">14.4.1.8/14-19</a>
0x00E	PCI Express Header Type Register	R	0x00 (EP mode) 0x01 (RC mode)	<a href="#">14.4.1.9/14-20</a>
0x00F	PCI Express BIST Register	R	0x00	<a href="#">14.4.1.10/14-21</a>
0x010– 0x014	Base Address Registers 0 and 1 (BAR0/BAR1) (EP mode only)	Mixed	0x0008	<a href="#">14.4.2.1.1/14-22</a>
0x018– 0x020	Base Address Registers 2 and 4 (BAR2/BAR4) (EP mode only)	Mixed	0x0000_000C	<a href="#">14.4.2.1.2/14-23</a>
0x01C– 0x024	Base Address Registers 3 and 5 (BAR3/BAR5) (EP mode only)	R/W	0x0000_0000	<a href="#">14.4.2.1.3/14-23</a>
0x02C	PCI Express Subsystem Vendor ID Register (EP mode only)	Special	0x0000	<a href="#">14.4.2.2/14-24</a>
0x02E	PCI Express Subsystem ID Register (EP mode only)	Special	0x0000	<a href="#">14.4.2.3/14-24</a>
0x034	PCI Express Capabilities Pointer Register	R	0x0044	<a href="#">14.4.2.4/14-25</a>
0x03C	PCI Express Interrupt Line Register (EP mode only)	R/W	0x0000	<a href="#">14.4.2.5/14-25</a>
0x03D	Reserved	—	—	—
0x03E	PCI Express Minimum Grant Register (EP mode only)	R	0x0000	<a href="#">14.4.2.6/14-26</a>
0x03F	PCI Express Maximum Latency Register (EP mode only)	R	0x0000	<a href="#">14.4.2.7/14-26</a>
0x018	PCI Express Primary Bus Number Register (RC mode only)	R/W	0x0000	<a href="#">14.4.3.1/14-27</a>
0x019	PCI Express Secondary Bus Number Register (RC mode only)	R/W	0x0000	<a href="#">14.4.3.2/14-28</a>
0x01A	PCI Express Subordinate Bus Number Register (RC mode only)	R/W	0x0000	<a href="#">14.4.3.3/14-28</a>
0x01B	Secondary Latency Timer Register 2 (RC mode only)	RO	0x0000	<a href="#">14.4.3.4/14-28</a>
0x01C	PCI Express I/O Base Register (RC mode only)	R	0x0000	<a href="#">14.4.3.5/14-29</a>
0x01D	PCI Express I/O Limit Register (RC mode only)	R	0x0000	<a href="#">14.4.3.6/14-29</a>
0x01E	PCI Express Secondary Status Register (RC mode only)	Mixed	0x0000	<a href="#">14.4.3.7/14-30</a>
0x020	PCI Express Memory Base Register (RC mode only) <sup>1</sup>	R/W	0x0000	<a href="#">14.4.3.8/14-30</a>

Table A-20. PCI Express Controller Registers

PCI Express—Block Base Address 0x0_9000				
Offset	Register	Access	Reset	Section/Page
0x022	PCI Express Memory Limit Register (RC mode only) <sup>1</sup>	R/W	0x0000	14.4.3.9/14-31
0x024	PCI Express Prefetchable Memory Base Register (RC mode only) <sup>1</sup>	R/W	0x0000	14.4.3.10/14-31
0x026	PCI Express Prefetchable Memory Limit Register (RC mode only) <sup>1</sup>	R/W	0x0000	14.4.3.11/14-32
0x028	PCI Express Prefetchable Base Upper 32-Bit Register (RC mode only) <sup>1</sup>	R/W	0x0000	14.4.3.12/14-32
0x02C	PCI Express Prefetchable Limit Upper 32-Bit Register (RC mode only) <sup>1</sup>	R/W	0x0000	14.4.3.13/14-33
0x030	PCI Express I/O Base Upper 16-Bit Register (RC mode only) <sup>1</sup>	R	0x0000	14.4.3.14/14-33
0x032	PCI Express I/O Limit Upper 16-Bit Register (RC mode only) <sup>1</sup>	R	0x0000	14.4.3.15/14-34
0x034	PCI Express Capabilities Pointer Register	R	0x044	14.4.3.16/14-34
0x03C	PCI Express Interrupt Line Register	R/W	0x0000	14.4.3.17/14-35
0x03D	Reserved	—	—	—
0x03E	PCI Express Bridge Control Register (RC mode only)	R/W	0x0000	14.4.3.18/14-35
0x044	PCI Express Power Management Capability ID Register	R	0x01	14.4.4.1/14-37
0x045	PCI Express Power Management Next Capabilities Pointer Register	R	0x4C	14.4.4.2/14-37
0x046	PCI Express Power Management Capabilities Register	R	0x7E02	14.4.4.3/14-38
0x048	PCI Express Power Management Status and Control Register	Mixed	0x0000	14.4.4.4/14-38
0x04B	PCI Express Power Management Data Register	R	0x0000	14.4.4.5/14-39
0x04C	PCI Express Capability ID Register	R	0x10	14.4.4.6/14-39
0x04D	PCI Express Next Capabilities Pointer Register	R	0x70	14.4.4.7/14-40
0x04E	PCI Express Capabilities Register	R	0x00n1	14.4.4.8/14-40
0x050	PCI Express Device Capabilities Register	R	0x0000_0000	14.4.4.9/14-41
0x054	PCI Express Device Control Register	R/W	0x2810	14.4.4.10/14-42
0x056	PCI Express Device Status Register	Mixed	0x0000	14.4.4.11/14-43
0x058	PCI Express Link Capabilities Register	R	0x0003_D411	14.4.4.12/14-43
0x05C	PCI Express Link Control Register	R/W	0x0000	14.4.4.13/14-44
0x05E	PCI Express Link Status Register	R	0x0011	14.4.4.14/14-45
0x060	PCI Express Slot Capabilities Register	R	0x000007c0	14.4.4.15/14-45
0x064	PCI Express Slot Control Register	R/W	0x0000	14.4.4.16/14-46
0x066	PCI Express Slot Status Register	Mixed	0x0040	14.4.4.17/14-47
0x068	PCI Express Root Control Register (RC mode only)	R/W	0x0000	14.4.4.18/14-47
0x06C	PCI Express Root Status Register (RC mode only)	Mixed	0x0000_0000	14.4.4.19/14-48

Table A-20. PCI Express Controller Registers

PCI Express—Block Base Address 0x0_9000				
Offset	Register	Access	Reset	Section/Page
0x070	PCI Express MSI Message Capability ID Register (EP mode only)	R	0x05	14.4.4.20/14-49
0x072	PCI Express MSI Message Control Register (EP mode only)	Mixed	0x0088	14.4.4.21/14-49
0x074	PCI Express MSI Message Address Register (EP mode only)	R/W	0x0000_0000	14.4.4.22/14-50
0x078	PCI Express MSI Message Upper Address Register (EP mode only)	R/W	0x0000_0000	14.4.4.23/14-50
0x07C	PCI Express MSI Message Data Register (EP mode only)	R/W	0x0000	14.4.4.24/14-50
0x100	PCI Express Advanced Error Reporting Capability ID Register	R	0x1381_0001	14.4.5.1/14-53
0x104	PCI Express Uncorrectable Error Status Register	R/W	0x0000_0000	14.4.5.2/14-53
0x108	PCI Express Uncorrectable Error Mask Register	R/W	0x0000_0000	14.4.5.3/14-54
0x10C	PCI Express Uncorrectable Error Severity Register	R/W	0x0006_2010	14.4.5.4/14-55
0x110	PCI Express Correctable Error Status Register	w1c	0x0000_0000	14.4.5.5/14-56
0x114	PCI Express Correctable Error Mask Register	R/W	0x0000_0000	14.4.5.6/14-57
0x118	PCI Express Advanced Error Capabilities and Control Register	R/W	0x0000_00A0	14.4.5.7/14-58
0x11C	PCI Express Header Log Register	R	0x0000_0000	14.4.5.8/14-58
0x120	PCI Express Header Log Register	R	0x0000_0000	
0x124	PCI Express Header Log Register	R	0x0000_0000	
0x128	PCI Express Header Log Register	R	0x0000_0000	
0x12C	PCI Express Root Error Command Register	R/W	0x0000_0000	14.4.5.9/14-60
0x130	PCI Express Root Error Status Register	Mixed	0x0000_0000	14.4.5.10/14-60
0x134	PCI Express Error Source Identification Register	R	0x0000_0000	14.4.5.11/14-61
PCI Express Core Control and Status Registers (CSRs)				
0x404	PCI Express LTSSM State Status Register (PEX_LTSSM_STAT)	R	0x0000_0000	14.4.6.1/14-62
0x41C	PCI Express N_FTS Control Register (PEX_NFTS_CTRL)	R/W	0x0000_4040	14.4.6.2/14-63
0x438	PCI Express ACK Replay Timeout Register (PEX_ACKRPLY_TO)	R/W	0x003B_2090	14.4.6.3/14-64
0x440	PCI Express Core Clock Ratio Register (PEX_GCLK_RATIO)	Mixed	0x0000_0010	14.4.6.4/14-65
0x450	PCI Express Power Management Timer Register (PEX_PM_TIMER)	Mixed	0x000A_63E4	14.4.6.5/14-66
0x454	PCI Express PME Time-Out Register (PEX_PME_TIMEOUT)	Mixed	0x00FD_4BC0	14.4.6.6/14-67
0x45C	PCI Express ASPM Request Timer Register (PEX_ASPM_REQTMR) (RC mode only)	R/W	0x0000_0629	14.4.6.7/14-67
0x478	PCI Express Subsystem Vendor ID Update Register (PEX_SSVID_UPDATE)	R/W	0x0000_0000	14.4.6.8/14-68
0x47C	PCI Express Device Capabilities Update Register (PEX_DEVCAP_UPDATE)	R/W	0x0000_0000	14.4.6.9/14-68



Table A-20. PCI Express Controller Registers

PCI Express—Block Base Address 0x0_9000				
Offset	Register	Access	Reset	Section/Page
0x480	PCI Express Link Capabilities Update Register (PEX_LINKCAP_UPDATE)	R/W	0x0000_3D41	14.4.6.10/14-69
0x490	PCI Express Slot Capabilities Update Register (PEX_SLCAP_UPDATE)	R/W	0x0000_07C0	14.4.6.11/14-71
0x4B0	PCI Express Configuration Ready Register (PEX_CFG_READY)	Mixed	0x0000_0000	14.4.6.12/14-71
PCI Express BAR Configuration Registers (EP Mode)				
0x4D8	PCI Express BAR Size Low Configuration Register (PEX_BAR_SIZEL)	R/W	0xFC00_0000	14.4.7.1/14-72
0x4DC	Reserved	—	—	—
0x4E0	PCI Express BAR Select Configuration Register (PEX_BAR_SEL)	R/W	0x0000_0400	14.4.7.2/14-73
0x504	PCI Express BAR Prefetch Configuration Register (PEX_BAR_PF)	R/W	0x0000_0400	14.4.7.3/14-73
PCI Express Extended Status and Control Register				
0x590	PCI Express PME_To_Ack Timeout Register (PEX_PME_TO_ACK_TOR)	Mixed	0x0019_5460	14.4.8.1/14-74
0x594	PCI Express PME_To_Ack Status Register (PEX_PME_TO_ACK_SR)	w1c	0x0000_0000	14.4.8.2/14-75
0x5A0	PCI Express PCI Interrupt Mask Register (PEX_SS_INTR_MASK)	Mixed	0x0000_003F	14.4.8.3/14-76
PCI Express CSB Bridge Registers				
Global Registers				
0x800–0x804	Reserved	—	—	—
0x808	PCI Express CSB Bridge Control register (PEX_CSB_CTRL)	R/W	0x0000_0130	14.5.2.1/14-77
0x80C	Reserved	—	—	—
0x814	PCI Express DMA Descriptor Timer Register (PEX_DMA_DSTMR)	R/W	0x0000_0000	14.5.2.2/14-78
0x818	Reserved	—	—	—
0x81C	PCI Express CSB Bridge Status register (PEX_CSB_STAT)	RO	0x0000_0000	14.5.2.3/14-79
0x820	Reserved	—	—	—
PCI Express Outbound PIO Registers				
0x840	PCI Express Outbound PIO Control Register (PEX_CSB_OBCTRL)	R/W	0x0000_0000	14.5.3.1/14-80
0x844	PCI Express Outbound PIO Status Register (PEX_CSB_OBSTAT)	w1c	0x0000_0000	14.5.3.2/14-81
0x848	Reserved	—	—	—
PCI Express Inbound PIO Registers				

Table A-20. PCI Express Controller Registers

PCI Express—Block Base Address 0x0_9000				
Offset	Register	Access	Reset	Section/Page
0x8E0	PCI Express Inbound PIO Control Register (PEX_CSB_IBCTRL)	R/W	0x0000_0000	14.5.4.1/14-82
0x8E4	PCI Express Inbound PIO Status Register (PEX_CSB_IBSTAT)	w1c	0x0000_0000	14.5.4.2/14-82
0x8E8	Reserved	—	—	—
PCI Express DMA Registers				
0x990	Reserved	—	—	—
0x9A0	PCI Express Write DMA Control Register (PEX_WDMA_CTRL)	R/W	0x0000_0000	14.5.5.1/14-83
0x9A4	PCI Express Write DMA first Address Register (PEX_WDMA_ADDR)	R/W	0x0000_0000	14.5.5.2/14-84
0x9A8	PCI Express Write DMA Status Register (PEX_WDMA_STAT)	w1c	0x0000_0000	14.5.5.3/14-84
0x9AC	Reserved	—	—	—
0xA40	PCI Express Read DMA Control Register (PEX_RDMA_CTRL)	R/W	0x0000_0000	14.5.5.4/14-85
0xA44	PCI Express Read DMA first Address Register (PEX_RDMA_ADDR)	R/W	0x0000_0000	14.5.5.5/14-86
0xA48	PCI Express Read DMA Status Register (PEX_RDMA_STAT)	w1c	0x0000_0000	14.5.5.6/14-86
Mailbox Registers				
0xB20	PCI Express Outbound Mailbox Control Register (PEX_OMBCR)	R/W	0x0000_0000	14.5.6.1/14-87
0xB24	PCI Express Outbound Mailbox Data Register (PEX_OMBDR)	R/W	0x0000_0000	14.5.6.2/14-88
0xB60	PCI Express Inbound Mailbox Control Register (PEX_IMBCR)	R/W	0x0000_0000	14.5.6.3/14-88
0xB64	PCI Express Inbound Mailbox Data Register (PEX_IMBDR)	R/W	0x0000_0000	14.5.6.4/14-89
PCI Express Host Interrupts Registers				
0xBA0	PCI Express Host Interrupt Enable Register (PEX_HIER)	R/W	0x0000_0000	14.5.7.1/14-89
0xBA4	PCI Express Host Interrupt Status Register (PEX_HISR)	w1c	0x0000_0000	14.5.7.2/14-90
0xBA8	PCI Express Host Outbound PIO Interrupt Vector Register (PEX_HOPIVR)	R/W	0x0000_0000	14.5.7.3/14-91
0xBC0	PCI Express Host Inbound PIO Interrupt Vector Register (PEX_HIPIVR)	R/W	0x0000_0000	14.5.7.4/14-92
0xBC8	PCI Express Host Write DMA Interrupt Vector Register (PEX_HWDIVR)	R/W	0x0000_0000	14.5.7.5/14-92
0xBD0	PCI Express Host Read DMA Interrupt Vector Register (PEX_HRDIVR)	R/W	0x0000_0000	14.5.7.6/14-93
0xBD8	PCI Express Host Miscellaneous Interrupt Vector Register (PEX_HMIVR)	R/W	0x0000_0000	14.5.7.7/14-93
CSB System Interrupts Registers				
0xBE0	CSB System PIO Interrupt Enable Register (PEX_CSPIER)	R/W	0x0000_0000	14.5.8.1/14-94

Table A-20. PCI Express Controller Registers

PCI Express—Block Base Address 0x0_9000				
Offset	Register	Access	Reset	Section/Page
0xBE4	CSB System Write DMA Interrupt Enable Register (PEX_CSWDIER)	R/W	0x0000_0000	14.5.8.2/14-94
0xBE8	CSB System Read DMA Interrupt Enable Register (PEX_CSRDIER)	R/W	0x0000_0000	14.5.8.3/14-95
0xBEC	CSB System Miscellaneous Interrupt Enable Register (PEX_CSMIER)	R/W	0x0000_0002	14.5.8.4/14-96
0xBF0	CSB System PIO Interrupt Status Register (PEX_CSPIISR)	w1c	0x0000_0000	14.5.8.5/14-98
0xBF4	CSB System Write DMA Interrupt Status Register (PEX_CSWDISR)	w1c	0x0000_0000	14.5.8.6/14-99
0xBF8	CSB System Read DMA Interrupt Status Register (PEX_CSRDISR)	w1c	0x0000_0000	14.5.8.7/14-99
0xBFC	CSB System Miscellaneous Interrupt Status Register (PEX_CSMISR)	w1c	0x0000_0000	14.5.8.8/14-100
Power Management Registers				
0xC80	PCI Express PM Control Register (PEX_PM_CTRL)	R/W	0x0000_0000	14.5.9.1/14-102
PCI Express Outbound Address Mapping Registers				
0xCA0	PCI Express Outbound Window Attributes Register 0 (PEX_OWAR0)	R/W	0x0000_0000	14.5.10.1/14-103
0xCA4	PCI Express Outbound Window Base Address Register 0 (PEX_OWBAR0)	R/W	0x0000_0000	14.5.10.2/14-104
0xCA8	PCI Express Outbound Window Translation Address Register Low 0 (PEX_OWTARL0)	R/W	0x0000_0000	14.5.10.3/14-105
0xCAC	PCI Express Outbound Window Translation Address Register High 0 (PEX_OWTARH0)	R/W	0x0000_0000	14.5.10.4/14-105
0xCB0	PCI Express Outbound Window Attributes Register 1 (PEX_OWAR1)	R/W	0x0000_0000	14.5.10.1/14-103
0xCB4	PCI Express Outbound Window Base Address Register 1 (PEX_OWBAR1)	R/W	0x0000_0000	14.5.10.2/14-104
0xCB8	PCI Express Outbound Window Translation Address Register Low 1 (PEX_OWTARL1)	R/W	0x0000_0000	14.5.10.3/14-105
0xCBC	PCI Express Outbound Window Translation Address Register High 1 (PEX_OWTARH1)	R/W	0x0000_0000	14.5.10.4/14-105
0xCC0	PCI Express Outbound Window Attributes Register 2 (PEX_OWAR2)	R/W	0x0000_0000	14.5.10.1/14-103
0xCC4	PCI Express Outbound Window Base Address Register 2 (PEX_OWBAR2)	R/W	0x0000_0000	14.5.10.2/14-104
0xCC8	PCI Express Outbound Window Translation Address Register Low 2 (PEX_OWTARL2)	R/W	0x0000_0000	14.5.10.3/14-105
0xCCC	PCI Express Outbound Window Translation Address Register High 2 (PEX_OWTARH2)	R/W	0x0000_0000	14.5.10.4/14-105

Table A-20. PCI Express Controller Registers

PCI Express—Block Base Address 0x0_9000				
Offset	Register	Access	Reset	Section/Page
0xCD0	PCI Express Outbound Window Attributes Register 3 (PEX_OWAR3)	R/W	0x0000_0000	14.5.10.1/14-103
0xCD4	PCI Express Outbound Window Base Address Register 3 (PEX_OWBAR3)	R/W	0x0000_0000	14.5.10.2/14-104
0xCD8	PCI Express Outbound Window Translation Address Register Low 3 (PEX_OWTLARL3)	R/W	0x0000_0000	14.5.10.3/14-105
0xCDC	PCI Express Outbound Window Translation Address Register High 3 (PEX_OWTLARH3)	R/W	0x0000_0000	14.5.10.4/14-105
PCI Express EP Inbound Address Translation Registers				
0xDE0	PCI Express EP Inbound Window Translation Address Register 0 (PEX_EPIWTAR0)	R/W	0x0000_0000	14.5.11.1/14-107
0xDE4	PCI Express EP Inbound Window Translation Address Register 1 (PEX_EPIWTAR1)	R/W	0x0000_0000	14.5.11.1/14-107
0xDE8	PCI Express EP Inbound Window Translation Address Register 2 (PEX_EPIWTAR2)	R/W	0x0000_0000	14.5.11.1/14-107
0xDEC	PCI Express EP Inbound Window Translation Address Register 3 (PEX_EPIWTAR3)	R/W	0x0000_0000	14.5.11.1/14-107
PCI Express RC Inbound Address Mapping Registers				
0xE60	PCI Express RC Inbound Window Attributes Register 0 (PEX_RCIWAR0)	R/W	0x0000_0000	14.5.12.1/14-108
0xE64	PCI Express RC Inbound Window Translation Address Register 0 (PEX_RCIWTAR0)	R/W	0x0000_0000	14.5.12.2/14-109
0xE68	PCI Express RC Inbound Window Base Address Register Low 0 (PEX_RCIWBARL0)	R/W	0x0000_0000	14.5.12.3/14-109
0xE6C	PCI Express RC Inbound Window Base Address Register High 0 (PEX_RCIWBARH0)	R/W	0x0000_0000	14.5.12.4/14-110
0xE70	PCI Express RC Inbound Window Attributes Register 1 (PEX_RCIWAR1)	R/W	0x0000_0000	14.5.12.1/14-108
0xE74	PCI Express RC Inbound Window Translation Address Register 1 (PEX_RCIWTAR1)	R/W	0x0000_0000	14.5.12.2/14-109
0xE78	PCI Express RC Inbound Window Base Address Register Low 1 (PEX_RCIWBARL1)	R/W	0x0000_0000	14.5.12.3/14-109
0xE7C	PCI Express RC Inbound Window Base Address Register High 1 (PEX_RCIWBARH1)	R/W	0x0000_0000	14.5.12.4/14-110
0xE80	PCI Express RC Inbound Window Attributes Register 2 (PEX_RCIWAR2)	R/W	0x0000_0000	14.5.12.1/14-108
0xE84	PCI Express RC Inbound Window Translation Address Register 2 (PEX_RCIWTAR2)	R/W	0x0000_0000	14.5.12.2/14-109
0xE88	PCI Express RC Inbound Window Base Address Register Low 2 (PEX_RCIWBARL2)	R/W	0x0000_0000	14.5.12.3/14-109

Table A-20. PCI Express Controller Registers

PCI Express—Block Base Address 0x0_9000				
Offset	Register	Access	Reset	Section/Page
0xE8C	PCI Express RC Inbound Window Base Address Register High 2 (PEX_RCIWBARH2)	R/W	0x0000_0000	14.5.12.4/14-110
0xE90	PCI Express RC Inbound Window Attributes Register 3 (PEX_RCIWAR3)	R/W	0x0000_0000	14.5.12.1/14-108
0xE94	PCI Express RC Inbound Window Translation Address Register 3 (PEX_RCIWTAR3)	R/W	0x0000_0000	14.5.12.2/14-109
0xE98	PCI Express RC Inbound Window Base Address Register Low 3 (PEX_RCIWBARL3)	R/W	0x0000_0000	14.5.12.3/14-109
0xE9C	PCI Express RC Inbound Window Base Address Register High 3 (PEX_RCIWBARH3)	R/W	0x0000_0000	14.5.12.4/14-110

<sup>1</sup> MPC8308 does not support these registers in accordance with the PCIe specification. For more information, see *PCI Express Base Specification, March 28, 2005* (Page 357-358). These registers are mentioned here only for completeness. It is recommended not to change the reset values of these registers.

## A.20 Enhanced Three-Speed Ethernet Controllers (eTSECs)

Table A-21. Enhanced Three-Speed Ethernet Controllers (eTSECs) Registers

eTSEC 1—Block Base Address 0x2_4000 eTSEC 2—Block Base Address 0x2_5000				
eTSEC1 Offset	Name <sup>1</sup>	Access <sup>2</sup>	Reset	Section/Page
<b>eTSEC General Control and Status Registers</b>				
0x000	TSEC_ID*—Controller ID register	R	0x0124_0106	16.5.3.1.1/16-21
0x004	TSEC_ID2*—Controller ID register	R	0x0030_00F0	16.5.3.1.2/16-22
0x008–0x00C	Reserved	—	—	—
0x010	IEVENT—Interrupt event register	w1c	0x0000_0000	16.5.3.1.3/16-22
0x014	IMASK—Interrupt mask register	R/W	0x0000_0000	16.5.3.1.4/16-26
0x018	EDIS—Error disabled register	R/W	0x0000_0000	16.5.3.1.5/16-28
0x01C	Reserved	—	—	—
0x020	ECNTRL—Ethernet control register	R/W	0x0000_0000	16.5.3.1.6/16-30
0x024	Reserved	—	—	—
0x028	PTV—Pause time value register	R/W	0x0000_0000	16.5.3.1.7/16-31
0x02C	DMACTRL—DMA control register	R/W	0x0000_0000	16.5.3.1.8/16-32
0x030	TBIPA—TBI PHY address register	R/W	0x0000_0000	16.5.3.1.9/16-33
0x034–0x0FC	Reserved	—	—	—

Table A-21. Enhanced Three-Speed Ethernet Controllers (eTSECs) Registers (continued)

eTSEC 1—Block Base Address 0x2_4000 eTSEC 2—Block Base Address 0x2_5000				
eTSEC1 Offset	Name <sup>1</sup>	Access <sup>2</sup>	Reset	Section/Page
<b>eTSEC Transmit Control and Status Registers</b>				
0x100	TCTRL—Transmit control register	R/W	0x0000_0000	16.5.3.2.1/16-34
0x104	TSTAT—Transmit status register	w1c	0x0000_0000	16.5.3.2.2/16-36
0x108	DFVLAN*—Default VLAN control word	R/W	0x8100_0000	16.5.3.2.3/16-40
0x10C	Reserved	—	—	—
0x110	TXIC—Transmit interrupt coalescing register	R/W	0x0000_0000	16.5.3.2.4/16-41
0x114	TQUEUE*—Transmit queue control register	R/W	0x0000_8000	16.5.3.2.5/16-42
0x118– 0x13C	Reserved	—	—	—
0x140	TR03WT*—TxBD Rings 0–3 round-robin weightings	R/W	0x0000_0000	16.5.3.2.6/16-42
0x144	TR47WT*—TxBD Rings 4–7 round-robin weightings	R/W	0x0000_0000	16.5.3.2.7/16-43
0x148– 0x180	Reserved	—	—	—
0x184	TBPTR0—TxBD pointer for ring 0	R/W	0x0000_0000	16.5.3.2.8/16-44
0x188	Reserved	—	—	—
0x18C	TBPTR1*—TxBD pointer for ring 1	R/W	0x0000_0000	16.5.3.2.8/16-44
0x190	Reserved	—	—	—
0x194	TBPTR2*—TxBD pointer for ring 2	R/W	0x0000_0000	16.5.3.2.8/16-44
0x198	Reserved	—	—	—
0x19C	TBPTR3*—TxBD pointer for ring 3	R/W	0x0000_0000	16.5.3.2.8/16-44
0x1A0	Reserved	—	—	—
0x1A4	TBPTR4*—TxBD pointer for ring 4	R/W	0x0000_0000	16.5.3.2.8/16-44
0x1A8	Reserved	—	—	—
0x1AC	TBPTR5*—TxBD pointer for ring 5	R/W	0x0000_0000	16.5.3.2.8/16-44
0x1B0	Reserved	—	—	—
0x1B4	TBPTR6*—TxBD pointer for ring 6	R/W	0x0000_0000	16.5.3.2.8/16-44
0x1B8	Reserved	—	—	—
0x1BC	TBPTR7*—TxBD pointer for ring 7	R/W	0x0000_0000	16.5.3.2.8/16-44
0x1C0– 0x200	Reserved	—	—	—
0x204	TBASE0—TxBD base address of ring 0	R/W	0x0000_0000	16.5.3.2.9/16-45
0x208	Reserved	—	—	—
0x20C	TBASE1*—TxBD base address of ring 1	R/W	0x0000_0000	16.5.3.2.9/16-45

Table A-21. Enhanced Three-Speed Ethernet Controllers (eTSECs) Registers (continued)

eTSEC 1—Block Base Address 0x2_4000 eTSEC 2—Block Base Address 0x2_5000				
eTSEC1 Offset	Name <sup>1</sup>	Access <sup>2</sup>	Reset	Section/Page
0x210	Reserved	—	—	—
0x214	TBASE2*—TxBD base address of ring 2	R/W	0x0000_0000	16.5.3.2.9/16-45
0x218	Reserved	—	—	—
0x21C	TBASE3*—TxBD base address of ring 3	R/W	0x0000_0000	16.5.3.2.9/16-45
0x220	Reserved	—	—	—
0x224	TBASE4*—TxBD base address of ring 4	R/W	0x0000_0000	16.5.3.2.9/16-45
0x228	Reserved	—	—	—
0x22C	TBASE5*—TxBD base address of ring 5	R/W	0x0000_0000	16.5.3.2.9/16-45
0x230	Reserved	—	—	—
0x234	TBASE6*—TxBD base address of ring 6	R/W	0x0000_0000	16.5.3.2.9/16-45
0x238	Reserved	—	—	—
0x23C	TBASE7*—TxBD base address of ring 7	R/W	0x0000_0000	16.5.3.2.9/16-45
0x240– 0x27C	Reserved	—	—	—
0x280	TMR_TXTS1_ID* - Tx time stamp identification (set 1)	R/W	0x0000_0000	16.5.3.2.10/16-45
0x284	TMR_TXTS2_ID* - Tx time stamp identification (set 2)	R/W	0x0000_0000	16.5.3.2.10/16-45
0x288– 0x2BC	Reserved	—	—	—
0x2C0	TMR_TXTS1_H* - Tx time stamp high (set 1)	R/W	0x0000_0000	16.5.3.2.11/16-46
0x2C4	TMR_TXTS1_L* - Tx time stamp high (set 1)	R/W	0x0000_0000	16.5.3.2.11/16-46
0x2C8	TMR_TXTS2_H* - Tx time stamp high (set 2)	R/W	0x0000_0000	16.5.3.2.11/16-46
0x2CC	TMR_TXTS2_L* - Tx time stamp high (set 2)	R/W	0x0000_0000	16.5.3.2.11/16-46
0x2D0– 0x2FC	Reserved	—	—	—
eTSEC Receive Control and Status Registers				
0x300	RCTRL—Receive control register	R/W	0x0000_0000	16.5.3.3.1/16-46
0x304	RSTAT—Receive status register	w1c	0x0000_0000	16.5.3.3.2/16-48
0x308– 0x30C	Reserved	—	—	—
0x310	RXIC—Receive interrupt coalescing register	R/W	0x0000_0000	16.5.3.3.3/16-50
0x314	RQUEUE*—Receive queue control register.	R/W	0x0080_0080	16.5.3.3.4/16-51
0x318– 0x32C	Reserved	—	—	—
0x330	RBIFX*—Receive bit field extract control register	R/W	0x0000_0000	16.5.3.3.5/16-52

Table A-21. Enhanced Three-Speed Ethernet Controllers (eTSECs) Registers (continued)

eTSEC 1—Block Base Address 0x2_4000 eTSEC 2—Block Base Address 0x2_5000				
eTSEC1 Offset	Name <sup>1</sup>	Access <sup>2</sup>	Reset	Section/Page
0x334	RQFAR*—Receive queue filing table address register	R/W	0x0000_0000	<a href="#">16.5.3.3.6/16-53</a>
0x338	RQFCR*—Receive queue filing table control register	R/W	0xn <sup>nnn</sup> _n <sup>nnn</sup>	<a href="#">16.5.3.3.7/16-54</a>
0x33C	RQFPR*—Receive queue filing table property register	R/W	0xn <sup>nnn</sup> _n <sup>nnn</sup>	<a href="#">16.5.3.3.8/16-55</a>
0x340	MRBLR—Maximum receive buffer length register	R/W	0x0000_0000	<a href="#">16.5.3.3.9/16-59</a>
0x344– 0x380	Reserved	—	—	—
0x384	RBPTR0—RxBd pointer for ring 0	R/W	0x0000_0000	<a href="#">16.5.3.3.10/16-60</a>
0x388	Reserved	—	—	—
0x38C	RBPTR1*—RxBd pointer for ring 1	R/W	0x0000_0000	<a href="#">16.5.3.3.10/16-60</a>
0x390	Reserved	—	—	—
0x394	RBPTR2*—RxBd pointer for ring 2	R/W	0x0000_0000	<a href="#">16.5.3.3.10/16-60</a>
0x398	Reserved	—	—	—
0x39C	RBPTR3*—RxBd pointer for ring 3	R/W	0x0000_0000	<a href="#">16.5.3.3.10/16-60</a>
0x3A0	Reserved	—	—	—
0x3A4	RBPTR4*—RxBd pointer for ring 4	R/W	0x0000_0000	<a href="#">16.5.3.3.10/16-60</a>
0x3A8	Reserved	—	—	—
0x3AC	RBPTR5*—RxBd pointer for ring 5	R/W	0x0000_0000	<a href="#">16.5.3.3.10/16-60</a>
0x3B0	Reserved	—	—	—
0x3B4	RBPTR6*—RxBd pointer for ring 6	R/W	0x0000_0000	<a href="#">16.5.3.3.10/16-60</a>
0x3B8	Reserved	—	—	—
0x3BC	RBPTR7*—RxBd pointer for ring 7	R/W	0x0000_0000	<a href="#">16.5.3.3.10/16-60</a>
0x3C0– 0x400	Reserved	—	—	—
0x404	RBASE0—RxBd base address of ring 0	R/W	0x0000_0000	<a href="#">16.5.3.3.11/16-60</a>
0x408	Reserved	—	—	—
0x40C	RBASE1*—RxBd base address of ring 1	R/W	0x0000_0000	<a href="#">16.5.3.3.11/16-60</a>
0x410	Reserved	—	—	—
0x414	RBASE2*—RxBd base address of ring 2	R/W	0x0000_0000	<a href="#">16.5.3.3.11/16-60</a>
0x418	Reserved	—	—	—
0x41C	RBASE3*—RxBd base address of ring 3	R/W	0x0000_0000	<a href="#">16.5.3.3.11/16-60</a>
0x420	Reserved	—	—	—
0x424	RBASE4*—RxBd base address of ring 4	R/W	0x0000_0000	<a href="#">16.5.3.3.11/16-60</a>
0x428	Reserved	—	—	—



Table A-21. Enhanced Three-Speed Ethernet Controllers (eTSECs) Registers (continued)

eTSEC 1—Block Base Address 0x2_4000 eTSEC 2—Block Base Address 0x2_5000				
eTSEC1 Offset	Name <sup>1</sup>	Access <sup>2</sup>	Reset	Section/Page
0x42C	RBASE5*—RxB D base address of ring 5	R/W	0x0000_0000	<a href="#">16.5.3.3.11/16-60</a>
0x430	Reserved	—	—	—
0x434	RBASE6*—RxB D base address of ring 6	R/W	0x0000_0000	<a href="#">16.5.3.3.11/16-60</a>
0x438	Reserved	—	—	—
0x43C	RBASE7*—RxB D base address of ring 7	R/W	0x0000_0000	<a href="#">16.5.3.3.11/16-60</a>
0x440–0x4BC	Reserved	—	—	—
0x4C0	TMR_RXTS_H* - Rx timer time stamp register high	R/W	0x0000_0000	<a href="#">16.5.3.3.12/16-61</a>
0x4C4	TMR_RXTS_L* - Rx timer time stamp register low	R/W	0x0000_0000	<a href="#">16.5.3.3.12/16-61</a>
0x4C8–0x4FC	Reserved	—	—	—
eTSEC MAC Registers				
0x500	MACCFG1—MAC configuration register 1	R/W	0x0000_0000	<a href="#">16.5.3.5.1/16-64</a>
0x504	MACCFG2—MAC configuration register 2	R/W	0x0000_7000	<a href="#">16.5.3.5.2/16-66</a>
0x508	IPGIFG—Inter-packet/inter-frame gap register	R/W	0x4060_5060	<a href="#">16.5.3.5.3/16-68</a>
0x50C	HAFDUP—Half-duplex control	R/W	0x00A1_F037	<a href="#">16.5.3.5.4/16-69</a>
0x510	MAXFRM—Maximum frame length	R/W	0x0000_0600	<a href="#">16.5.3.5.5/16-70</a>
0x514–0x51C	Reserved	—	—	—
0x520	MIIMCFG—MII management configuration	R/W	0x0000_0007	<a href="#">16.5.3.5.6/16-70</a>
0x524	MIIMCOM—MII management command	R/W	0x0000_0000	<a href="#">16.5.3.5.7/16-71</a>
0x528	MIIMADD—MII management address	R/W	0x0000_0000	<a href="#">16.5.3.5.8/16-72</a>
0x52C	MIIMCON—MII management control	WO	0x0000_0000	<a href="#">16.5.3.5.9/16-72</a>
0x530	MIIMSTAT—MII management status	R	0x0000_0000	<a href="#">16.5.3.5.10/16-73</a>
0x534	MIIMIND—MII management indicator	R	0x0000_0000	<a href="#">16.5.3.5.11/16-73</a>
0x538	Reserved	—	—	—
0x53C	IFSTAT—Interface status	R	0x0000_0000	<a href="#">16.5.3.5.12/16-74</a>
0x540	MACSTNADDR1—MAC station address register 1	R/W	0x0000_0000	<a href="#">16.5.3.5.13/16-74</a>
0x544	MACSTNADDR2—MAC station address register 2	R/W	0x0000_0000	<a href="#">16.5.3.5.14/16-75</a>

Table A-21. Enhanced Three-Speed Ethernet Controllers (eTSECs) Registers (continued)

eTSEC 1—Block Base Address 0x2_4000 eTSEC 2—Block Base Address 0x2_5000					
eTSEC1 Offset	Name <sup>1</sup>	Access <sup>2</sup>	Reset	Section/Page	
0x548	MAC01ADDR1*—MAC exact match address 1, part 1	R/W	0x0000_0000	16.5.3.5.15/16-75 16.5.3.5.16/16-76	
0x54C	MAC01ADDR2*—MAC exact match address 1, part 2	R/W	0x0000_0000		
0x550	MAC02ADDR1*—MAC exact match address 2, part 1	R/W	0x0000_0000		
0x554	MAC02ADDR2*—MAC exact match address 2, part 2	R/W	0x0000_0000		
0x558	MAC03ADDR1*—MAC exact match address 3, part 1	R/W	0x0000_0000		
0x55C	MAC03ADDR2*—MAC exact match address 3, part 2	R/W	0x0000_0000		
0x560	MAC04ADDR1*—MAC exact match address 4, part 1	R/W	0x0000_0000		
0x564	MAC04ADDR2*—MAC exact match address 4, part 2	R/W	0x0000_0000		
0x568	MAC05ADDR1*—MAC exact match address 5, part 1	R/W	0x0000_0000		
0x56C	MAC05ADDR2*—MAC exact match address 5, part 2	R/W	0x0000_0000		
0x570	MAC06ADDR1*—MAC exact match address 6, part 1	R/W	0x0000_0000		16.5.3.5.15/16-75 16.5.3.5.16/16-76
0x574	MAC06ADDR2*—MAC exact match address 6, part 2	R/W	0x0000_0000		
0x578	MAC07ADDR1*—MAC exact match address 7, part 1	R/W	0x0000_0000		
0x57C	MAC07ADDR2*—MAC exact match address 7, part 2	R/W	0x0000_0000		
0x580	MAC08ADDR1*—MAC exact match address 8, part 1	R/W	0x0000_0000		
0x584	MAC08ADDR2*—MAC exact match address 8, part 2	R/W	0x0000_0000		
0x588	MAC09ADDR1*—MAC exact match address 9, part 1	R/W	0x0000_0000		
0x58C	MAC09ADDR2*—MAC exact match address 9, part 2	R/W	0x0000_0000		
0x590	MAC10ADDR1*—MAC exact match address 10, part 1	R/W	0x0000_0000		
0x594	MAC10ADDR2*—MAC exact match address 10, part 2	R/W	0x0000_0000		
0x598	MAC11ADDR1*—MAC exact match address 11, part 1	R/W	0x0000_0000		
0x59C	MAC11ADDR2*—MAC exact match address 11, part 2	R/W	0x0000_0000		
0x5A0	MAC12ADDR1*—MAC exact match address 12, part 1	R/W	0x0000_0000		
0x5A4	MAC12ADDR2*—MAC exact match address 12, part 2	R/W	0x0000_0000		
0x5A8	MAC13ADDR1*—MAC exact match address 13, part 1	R/W	0x0000_0000		
0x5AC	MAC13ADDR2*—MAC exact match address 13, part 2	R/W	0x0000_0000		
0x5B0	MAC14ADDR1*—MAC exact match address 14, part 1	R/W	0x0000_0000		
0x5B4	MAC14ADDR2*—MAC exact match address 14, part 2	R/W	0x0000_0000		
0x5B8	MAC15ADDR1*—MAC exact match address 15, part 1	R/W	0x0000_0000		
0x5BC	MAC15ADDR2*—MAC exact match address 15, part 2	R/W	0x0000_0000		
0x5C0–0x67C	Reserved	—	—	—	
eTSEC Transmit and Receive Counters					
0x680	TR64—Transmit and receive 64-byte frame counter	R/W	0x0000_0000	16.5.3.6.1/16-78	

Table A-21. Enhanced Three-Speed Ethernet Controllers (eTSECs) Registers (continued)

eTSEC 1—Block Base Address 0x2_4000 eTSEC 2—Block Base Address 0x2_5000				
eTSEC1 Offset	Name <sup>1</sup>	Access <sup>2</sup>	Reset	Section/Page
0x684	TR127—Transmit and receive 65- to 127-byte frame counter	R/W	0x0000_0000	16.5.3.6.2/16-78
0x688	TR255—Transmit and receive 128- to 255-byte frame counter	R/W	0x0000_0000	16.5.3.6.3/16-79
0x68C	TR511—Transmit and receive 256- to 511-byte frame counter	R/W	0x0000_0000	16.5.3.6.4/16-79
0x690	TR1K—Transmit and receive 512- to 1023-byte frame counter	R/W	0x0000_0000	16.5.3.6.5/16-80
0x694	TRMAX—Transmit and receive 1024- to 1518-byte frame counter	R/W	0x0000_0000	16.5.3.6.6/16-80
0x698	TRMGV—Transmit and receive 1519- to 1522-byte good VLAN frame count	R/W	0x0000_0000	16.5.3.6.7/16-81
eTSEC Receive Counters				
0x69C	RBYT—Receive byte counter	R/W	0x0000_0000	16.5.3.6.8/16-81
0x6A0	RPKT—Receive packet counter	R/W	0x0000_0000	16.5.3.6.9/16-81
0x6A4	RFCS—Receive FCS error counter	R/W	0x0000_0000	16.5.3.6.10/16-82
0x6A8	RMCA—Receive multicast packet counter	R/W	0x0000_0000	16.5.3.6.11/16-82
0x6AC	RBCA—Receive broadcast packet counter	R/W	0x0000_0000	16.5.3.6.12/16-83
0x6B0	RXCF—Receive control frame packet counter	R/W	0x0000_0000	16.5.3.6.13/16-83
0x6B4	RXPF—Receive PAUSE frame packet counter	R/W	0x0000_0000	16.5.3.6.14/16-84
0x6B8	RXUO—Receive unknown OP code counter	R/W	0x0000_0000	16.5.3.6.15/16-84
0x6BC	RALN—Receive alignment error counter	R/W	0x0000_0000	16.5.3.6.16/16-85
0x6C0	RFLR—Receive frame length error counter	R/W	0x0000_0000	16.5.3.6.17/16-85
0x6C4	RCDE—Receive code error counter	R/W	0x0000_0000	16.5.3.6.18/16-86
0x6C8	RCSE—Receive carrier sense error counter	R/W	0x0000_0000	16.5.3.6.19/16-86
0x6CC	RUND—Receive undersize packet counter	R/W	0x0000_0000	16.5.3.6.20/16-87
0x6D0	ROVR—Receive oversize packet counter	R/W	0x0000_0000	16.5.3.6.21/16-87
0x6D4	RFRG—Receive fragments counter	R/W	0x0000_0000	16.5.3.6.22/16-88
0x6D8	RJBR—Receive jabber counter	R/W	0x0000_0000	16.5.3.6.23/16-88
0x6DC	RDRP—Receive drop counter	R/W	0x0000_0000	16.5.3.6.24/16-89
eTSEC Transmit Counters				
0x6E0	TBYT—Transmit byte counter	R/W	0x0000_0000	16.5.3.6.25/16-89
0x6E4	TPKT—Transmit packet counter	R/W	0x0000_0000	16.5.3.6.26/16-90
0x6E8	TMCA—Transmit multicast packet counter	R/W	0x0000_0000	16.5.3.6.27/16-90
0x6EC	TBCA—Transmit broadcast packet counter	R/W	0x0000_0000	16.5.3.6.28/16-91
0x6F0	TXPF—Transmit PAUSE control frame counter	R/W	0x0000_0000	16.5.3.6.29/16-91
0x6F4	TDFR—Transmit deferral packet counter	R/W	0x0000_0000	16.5.3.6.30/16-92
0x6F8	TEDF—Transmit excessive deferral packet counter	R/W	0x0000_0000	16.5.3.6.31/16-92

Table A-21. Enhanced Three-Speed Ethernet Controllers (eTSECs) Registers (continued)

eTSEC 1—Block Base Address 0x2_4000 eTSEC 2—Block Base Address 0x2_5000				
eTSEC1 Offset	Name <sup>1</sup>	Access <sup>2</sup>	Reset	Section/Page
0x6FC	TSCL—Transmit single collision packet counter	R/W	0x0000_0000	16.5.3.6.32/16-93
0x700	TMCL—Transmit multiple collision packet counter	R/W	0x0000_0000	16.5.3.6.33/16-93
0x704	TLCL—Transmit late collision packet counter	R/W	0x0000_0000	16.5.3.6.34/16-94
0x708	TXCL—Transmit excessive collision packet counter	R/W	0x0000_0000	16.5.3.6.35/16-94
0x70C	TNCL—Transmit total collision counter	R/W	0x0000_0000	16.5.3.6.36/16-95
0x710	Reserved	—	—	—
0x714	TDRP—Transmit drop frame counter	R/W	0x0000_0000	16.5.3.6.37/16-95
0x718	TJBR—Transmit jabber frame counter	R/W	0x0000_0000	16.5.3.6.38/16-96
0x71C	TFCS—Transmit FCS error counter	R/W	0x0000_0000	16.5.3.6.39/16-96
0x720	TXCF—Transmit control frame counter	R/W	0x0000_0000	16.5.3.6.40/16-97
0x724	TOVR—Transmit oversize frame counter	R/W	0x0000_0000	16.5.3.6.41/16-97
0x728	TUND—Transmit undersize frame counter	R/W	0x0000_0000	16.5.3.6.42/16-98
0x72C	TFRG—Transmit fragments frame counter	R/W	0x0000_0000	16.5.3.6.43/16-98
eTSEC Counter Control and TOE Statistics Registers				
0x730	CAR1—Carry register one register <sup>3</sup>	w1c	0x0000_0000	16.5.3.6.44/16-99
0x734	CAR2—Carry register two register <sup>3</sup>	w1c	0x0000_0000	16.5.3.6.45/16-100
0x738	CAM1—Carry register one mask register	R/W	0xFE03_FFFF	16.5.3.6.46/16-101
0x73C	CAM2—Carry register two mask register	R/W	0x000F_FFFD	16.5.3.6.47/16-103
0x740	RREJ*—Receive filer rejected packet counter	R/W	0x0000_0000	16.5.3.6.48/16-104
0x744– 0x7FC	Reserved	—	—	—
Hash Function Registers				
0x800	IGADDR0—Individual/group address register 0	R/W	0x0000_0000	16.5.3.7.1/16-105
0x804	IGADDR1—Individual/group address register 1	R/W	0x0000_0000	
0x808	IGADDR2—Individual/group address register 2	R/W	0x0000_0000	
0x80C	IGADDR3—Individual/group address register 3	R/W	0x0000_0000	
0x810	IGADDR4—Individual/group address register 4	R/W	0x0000_0000	
0x814	IGADDR5—Individual/group address register 5	R/W	0x0000_0000	
0x818	IGADDR6—Individual/group address register 6	R/W	0x0000_0000	
0x81C	IGADDR7—Individual/group address register 7	R/W	0x0000_0000	
0x820– 0x87C	Reserved	—	—	—

Table A-21. Enhanced Three-Speed Ethernet Controllers (eTSECs) Registers (continued)

eTSEC 1—Block Base Address 0x2_4000 eTSEC 2—Block Base Address 0x2_5000				
eTSEC1 Offset	Name <sup>1</sup>	Access <sup>2</sup>	Reset	Section/Page
0x880	GADDR0—Group address register 0	R/W	0x0000_0000	16.5.3.7.2/16-105
0x884	GADDR1—Group address register 1	R/W	0x0000_0000	
0x888	GADDR2—Group address register 2	R/W	0x0000_0000	
0x88C	GADDR3—Group address register 3	R/W	0x0000_0000	
0x890	GADDR4—Group address register 4	R/W	0x0000_0000	
0x894	GADDR5—Group address register 5	R/W	0x0000_0000	
0x898	GADDR6—Group address register 6	R/W	0x0000_0000	
0x89C	GADDR7—Group address register 7	R/W	0x0000_0000	
0x8A0–0xAFC	Reserved	—	—	—
eTSEC DMA Attribute Registers				
0xB00–0xBF4	Reserved	—	—	—
0xBF8	ATTR—Attribute register	R/W	0x0000_0000	16.5.3.8.1/16-106
0xBFC	ATTRELI*—Attribute extract length and extract index register	R/W	0x0000_0000	16.5.3.8.2/16-106
eTSEC Lossless Flow Control Registers				
0xC00	RQPRM0*—Receive Queue Parameters register 0	R/W	0x0000_0000	16.5.3.9.1/16-107
0xC04	RQPRM1*—Receive Queue Parameters register 1	R/W	0x0000_0000	
0xC08	RQPRM2*—Receive Queue Parameters register 2	R/W	0x0000_0000	
0xC0C	RQPRM3*—Receive Queue Parameters register 3	R/W	0x0000_0000	
0xC10	RQPRM4*—Receive Queue Parameters register 4	R/W	0x0000_0000	
0xC14	RQPRM5*—Receive Queue Parameters register 5	R/W	0x0000_0000	
0xC18	RQPRM6*—Receive Queue Parameters register 6	R/W	0x0000_0000	
0xC1C	RQPRM7*—Receive Queue Parameters register 7	R/W	0x0000_0000	
0xC20–0xC40	Reserved	—	—	—
0xC44	RFBPTR0*—Last Free RxBD pointer for ring 0	R/W	0x0000_0000	16.5.3.9.2/16-108
0xC48	Reserved	—	—	—
0xC4C	RFBPTR1*—Last Free RxBD pointer for ring 1	R/W	0x0000_0000	16.5.3.9.2/16-108
0xC50	Reserved	—	—	—
0xC54	RFBPTR2*—Last Free RxBD pointer for ring 2	R/W	0x0000_0000	16.5.3.9.2/16-108
0xC58	Reserved	—	—	—
0xC5C	RFBPTR3*—Last Free RxBD pointer for ring 3	R/W	0x0000_0000	16.5.3.9.2/16-108

Table A-21. Enhanced Three-Speed Ethernet Controllers (eTSECs) Registers (continued)

eTSEC 1—Block Base Address 0x2_4000 eTSEC 2—Block Base Address 0x2_5000				
eTSEC1 Offset	Name <sup>1</sup>	Access <sup>2</sup>	Reset	Section/Page
0xC60	Reserved	—	—	—
0xC64	RFBPTR4*—Last Free RxBD pointer for ring 4	R/W	0x0000_0000	16.5.3.9.2/16-108
0xC68	Reserved	—	—	—
0xC6C	RFBPTR5*—Last Free RxBD pointer for ring 5	R/W	0x0000_0000	16.5.3.9.2/16-108
0xC70	Reserved	—	—	—
0xC74	RFBPTR6*—Last Free RxBD pointer for ring 6	R/W	0x0000_0000	16.5.3.9.2/16-108
0xC78	Reserved	—	—	—
0xC7C	RFBPTR7*—Last Free RxBD pointer for ring 7	R/W	0x0000_0000	16.5.3.9.2/16-108
eTSEC Future Expansion Space				
0xCC0–0xD94	Reserved	—	—	—
eTSEC IEEE 1588 Registers				
0xE00	TMR_CTRL* - Timer control register	R/W	0x0001_0001	16.5.3.10.1/16-109
0xE04	TMR_TEVENT* - time stamp event register	w1c	0x0000_0000	16.5.3.10.2/16-111
0xE08	TMR_TEMASK* - Timer event mask register	R/W	0x0000_0000	16.5.3.10.3/16-113
0xE0C	TMR_PEVENT* - time stamp event register	R/W	0x0000_0000	16.5.3.10.4/16-113
0xE10	TMR_PEMASK* - Timer event mask register	R/W	0x0000_0000	16.5.3.10.5/16-114
0xE14	TMR_STAT* - time stamp status register	R/W	0x0000_0000	16.5.3.10.6/16-115
0xE18	TMR_CNT_H* - timer counter high register	R/W	0x0000_0000	16.5.3.10.7/16-115
0xE1C	TMR_CNT_L* - timer counter low register	R/W	0x0000_0000	16.5.3.10.7/16-115
0xE20	TMR_ADD* - Timer drift compensation addend register	R/W	0x0000_0000	16.5.3.10.8/16-116
0xE24	TMR_ACC* - Timer accumulator register	R/W	0x0000_0000	16.5.3.10.9/16-117
0xE28	TMR_PRSC* -Timer prescale	R/W	0x0000_0002	16.5.3.10.10/16-117
0xE2C	Reserved	—	—	—
0xE30	TMROFF_H* - Timer offset high	R/W	0x0000_0000	16.5.3.10.11/16-118
0xE34	TMROFF_L* - Timer offset low	R/W	0x0000_0000	16.5.3.10.11/16-118
0xE40	TMR_ALARM1_H* - Timer alarm 1 high register	R/W	0xFFFF_FFFF	16.5.3.10.12/16-118
0xE44	TMR_ALARM1_L* - Timer alarm 1 high register	R/W	0xFFFF_FFFF	
0xE48	TMR_ALARM2_H* - Timer alarm 2 high register	R/W	0xFFFF_FFFF	
0xE4C	TMR_ALARM2_L* - Timer alarm 2 high register	R/W	0xFFFF_FFFF	
0xE50–0xE7C	Reserved	—	—	—

Table A-21. Enhanced Three-Speed Ethernet Controllers (eTSECs) Registers (continued)

eTSEC 1—Block Base Address 0x2_4000 eTSEC 2—Block Base Address 0x2_5000				
eTSEC1 Offset	Name <sup>1</sup>	Access <sup>2</sup>	Reset	Section/Page
0xE80	TMR_FIPER1* - Timer fixed period interval	R/W	0xFFFF_FFFF	16.5.3.10.13/16-119
0xE84	TMR_FIPER2* - Timer fixed period interval	R/W	0xFFFF_FFFF	
0xE88	TMR_FIPER*3 - Timer fixed period interval	R/W	0xFFFF_FFFF	
0xEA0	TMR_ETTS1_H* - Time stamp of general purpose external trigger	R/W	0x0000_0000	16.5.3.10.14/16-120
0xEA4	TMR_ETTS1_L* - Time stamp of general purpose external trigger	R/W	0x0000_0000	
0xEA8	TMR_ETTS2_H* - Time stamp of general purpose external trigger	R/W	0x0000_0000	
0xEAC	TMR_ETTS2_L* - Time stamp of general purpose external trigger	R/W	0x0000_0000	
0xEB0–0xFFFF	Reserved	—	—	—
Other eTSECs				
0x000–0xFFFF	eTSEC2 REGISTERS <sup>4</sup>			

<sup>1</sup> Registers denoted \* are new to the enhanced TSEC and not supported by PowerQUICC II Pro TSECs.

<sup>2</sup> Key: R = read only, WO = write only, R/W = read and write, LH = latches high, SC = self-clearing.

<sup>3</sup> Cleared on read.

<sup>4</sup> eTSEC2 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\_5000 to 0x2\_5FFF.

## A.21 SerDes PHY

Table A-22. SerDes PHY Registers

SerDes PHY—Block Base Address 0xE_3000				
Offset	Register	Access	Reset	Section/Page
0x000	SRDSCR0—SerDes Control Register 0	R/W	0x1100_CC30	15.3.1/15-4
0x004	SRDSCR1—SerDes Control Register 1	R/W	0x0000_0040	15.3.2/15-6
0x008	SRDSCR2—SerDes Control Register 2	R/W	0x0080_0000	15.3.3/15-7
0x00C	SRDSCR3—SerDes Control Register 3	R/W	0x0101_0000	15.3.4/15-8
0x010	SRDSCR4—SerDes Control Register 4	R/W	0xnn00_0n0n	15.3.5/15-9
0x014–0x01C	Reserved	—	—	—
0x020	SRDSRSTCTL—SerDes Reset Control Register	R/W	0x0044_4500	15.3.6/15-10
0x024–0x1FC	Reserved	—	—	—

## A.22 Enhanced Secure Digital Host Controller (eSDHC)

Table A-23. Enhanced Secure Digital Host Controller (eSDHC) Registers

eSDHC Registers—Block Base Address 0x2_E000				
Offset	Register	Access	Reset	Section/Page
0x000	DMA system address (DSADDR)	R/W	0x0000_0000	11.4.1/11-6
0x004	Block attributes (BLKATTR)	R/W	0x0001_0000	11.4.2/11-6
0x008	Command argument (CMDARG)	R/W	0x0000_0000	11.4.3/11-7
0x00C	Command transfer type (XFERTYP)	R/W	0x0000_0000	11.4.4/11-8
0x010	Command response0 (CMDRSP0)	R	0x0000_0000	11.4.5/11-11
0x014	Command response1 (CMDRSP1)	R	0x0000_0000	11.4.5/11-11
0x018	Command response2 (CMDRSP2)	R	0x0000_0000	11.4.5/11-11
0x01C	Command response3 (CMDRSP3)	R	0x0000_0000	11.4.5/11-11
0x020	Data buffer access port (DATPORT)	R/W	0x0000_0000	11.4.6/11-13
0x024	Present state (PRSTAT)	R	0x0F80_00F8	11.4.7/11-13
0x028	Protocol control (PROCTL)	R/W	0x0000_0020	11.4.8/11-17
0x02C	System control (SYSCTL)	Mixed	0x0000_8008	11.4.9/11-20
0x030	Interrupt status (IRQSTAT)	w1c	0x0000_0000	11.4.10/11-23
0x034	Interrupt status enable (IRQSTATEN)	R/W	0x117F_013F	11.4.11/11-27
0x038	Interrupt signal enable (IRQSIGEN)	R/W	0x0000_0000	11.4.12/11-30
0x03C	Auto CMD12 status (AUTOC12ERR)	R	0x0000_0000	11.4.13/11-32
0x040	Host controller capabilities (HOSTCAPBLT)	R	0x01F3_0000	11.4.14/11-34
0x044 <sup>1</sup>	Watermark level (WML)	R/W	0x1010_1010	11.4.15/11-35
0x050	Force event (FEVT)	W	0x0000_0000	11.4.16/11-35
0x0FC	Host controller version (HOSTVER)	R	0x0000_1201	11.4.17/11-37
0x40C	DMA control register (DCR)	R/W	0x0000_0000	11.4.18/11-37

<sup>1</sup> The addresses following 0x044, except 0x050, 0x0FC and 0x40C, are reserved and read as all 0s. Writes to these registers are ignored.

## A.23 Universal Serial Bus (USB) Interface

Table A-24. USB Interface Registers

Offset	Register	Access	Reset	Section/Page
<b>USB DR Controller Registers</b>				
<b>USB DR Controller—Block Base Address 0x2_3000</b>				
0x000–0x0FF	Reserved, should be cleared	—	—	—



Table A-24. USB Interface Registers (continued)

Offset	Register	Access	Reset	Section/Page
0x100	CAPLENGTH—Capability register length	R	0x40	13.3.1.1/1313-6
0x102	HCIVERSION—Host interface version number <sup>1</sup>	R	0x0100	13.3.1.2/1313-7
0x104	HCSPARAMS—Host controller structural parameters <sup>1</sup>	R	0x0001_0011	13.3.1.3/1313-7
0x108	HCCPARAMS—Host controller capability parameters <sup>1</sup>	R	0x0000_0006	13.3.1.4/1313-8
0x120	DCIVERSION—Device interface version number	R	0x0001	13.3.1.5/1313-9
0x124	DCCPARAMS—Device controller parameters	R	0x0000_0183	13.3.1.6/1313-10
0x140	USBCMD—USB command	Mixed	0x0008_0000	13.3.2.1/1313-10
0x144	USBSTS—USB status	Mixed	0x0000_0000	13.3.2.2/1313-13
0x148	USBINTR—USB interrupt enable	R/W	0x0000_0000	13.3.2.3/1313-15
0x14C	FRINDEX—USB frame index	R/W	0x0000_0000	13.3.2.4/1313-16
0x154	PERIODICLISTBASE—Frame list base address <sup>1</sup>	R/W	0x0000_0000	13.3.2.6/1313-18
	DEVICEADDR—USB device address	R/W	0x0000_0000	13.3.2.7/1313-18
0x158	ASYNCLISTADDR—Next asynchronous list addr (host mode) <sup>1</sup>	R/W	0x0000_0000	13.3.2.8/1313-19
	ENDPOINTLISTADDR—Address at endpoint list (device mode)	R/W	0x0000_0000	13.3.2.9/1313-20
0x160	BURSTSIZE—Programmable burst size	R/W	0x0000_1010	13.3.2.10/1313-20
0x164	TXFILLTUNING—Host TT transmit pre-buffer packet tuning	R/W	0x0000_0000	13.3.2.11/1313-21
0x170	ULPI VIEWPORT—ULPI Register Access	Mixed	0x0000_0000	13.3.2.12/1313-23
0x180	CONFIGFLAG—Configured flag register	R	0x0000_0001	13.3.2.13/1313-24
0x184	PORTSC—Port status/control	Mixed	0x1000_0000	13.3.2.14/1313-25
0x1A4	OTGSC—On-The-Go status and control <sup>1</sup>	Mixed	0x200C_0000	13.3.2.15/1313-30
0x1A8	USBMODE—USB device mode	R/W	0x0000_0000	13.3.2.16/1313-32
0x1AC	ENDPTSETUPSTAT—Endpoint setup status	R/W	0x0000_0000	13.3.2.17/1313-33
0x1B0	ENDPOINTPRIME—Endpoint initialization	R/W	0x0000_0000	13.3.2.18/1313-34
0x1B4	ENDPTFLUSH—Endpoint de-initialize	R/W	0x0000_0000	13.3.2.19/1313-35
0x1B8	ENDPTSTATUS—Endpoint status	R	0x0000_0000	13.3.2.20/1313-35
0x1BC	ENDPTCOMPLETE—Endpoint complete	w1c	0x0000_0000	13.3.2.21/1313-36
0x1C0	ENDPTCTRL0—Endpoint control 0	Mixed	0x0080_0080	13.3.2.22/1313-37
0x1C4	ENDPTCTRL1—Endpoint control 1	R/W	0x0000_0000	13.3.2.23/1313-38
0x1C8	ENDPTCTRL2—Endpoint control 2	R/W	0x0000_0000	13.3.2.23/1313-38
0x1CA–0x1D4	Reserved	—	—	—
0x400	SNOOP1—Snoop 1	R/W	0x0000_0000	13.3.2.24/1313-39
0x404	SNOOP2—Snoop 2	R/W	0x0000_0000	13.3.2.24/1313-39
0x408	AGE_CNT_THRESH—Age count threshold	R/W	0x0000_0000	13.3.2.25/1313-40

**Table A-24. USB Interface Registers (continued)**

Offset	Register	Access	Reset	Section/Page
0x40C	PRI_CTRL—Priority control	R/W	0x0000_0000	<a href="#">13.3.2.26/1313-42</a>
0x410	SI_CTRL—System interface control	R/W	0x0000_0000	<a href="#">13.3.2.27/1313-42</a>
0x500	CONTROL—Control	R/W	0x0000_0000	<a href="#">13.3.2.28/1313-43</a>
0x504– 0xFFFF	Reserved, should be cleared	—	—	—

<sup>1</sup> This register has separate functions for the host and device operation; the host function is listed first in the table.